Space and Missile Systems Center (SMC)

Software Acquisition Handbook

SMC/AXE

Prepared by:
Michael Zambrana
Dennis Singer

Version 1.0

9 February, 2004

This page intentionally left blank

Foreword

This Handbook was prepared for the United States Air Force Space and Missile Systems Center (SMC). It is intended as a both a primer and reference to software engineering for your use. It is not all-inclusive and should be supplemented with Air Force and Department of Defense (DoD) directives, policies and procedures – see http://deskbook.dau.mil/jsp/default.jsp.

Finally, this text should be considered only a starting point. The SMC environment is undergoing rapid evolution. Over the next few years, the SMC Systems Engineering Revitalization (SER) initiatives will undoubtedly induce many changes to the conduct of engineering and acquisitions at the Center.

The views found in this Handbook of the authors, as they are not endorsed by SMC.

Preface

This Software Acquisition Handbook is written to provide SMC personnel with fundamental software engineering concepts and techniques as they apply to space and launch systems and the SMC environment. The intended audience includes the project officer, junior software/system engineer, an engineer in another discipline that must perform software acquisition engineering functions, or the experienced engineer who needs a reference. The authors recognize that software engineering subject matter is very broad and that approaches to performing software engineering vary. This exposition is not intended to cover them all. However, the content of this Handbook and Appendix A Lessons Learned provide sufficient information and recommendations to keep any software intensive effort successfully on course and not turn it into the Death-March Project. It addresses general concepts and common processes, tools, and techniques that are mostly familiar to SMC, DoD and Industry. It also provides information on recommended software engineering practices and pitfalls (see Appendix A Lessons Learned) to avoid. Many references are provided for the reader to consult for more in-depth knowledge.

This handbook describes software engineering as it could be applied to the development of major space and launch systems. Software engineering provides a disciplined approach that covers the entire lifecycle of a system to include development, design, manufacture, and operation. Consequently, the handbook's scope properly includes software engineering functions regardless of whether they are performed by the AFSPC operational user, SMC system program office (Program Office), or a systems contractor. This book is also prepared to accommodate the SMC systems engineering training program.

Executive Summary

A Weapon System Software Sustainment Study completed by AFMC in 2001 identified lack of guidance as a major concern for Air Force organizations involved in the acquisition and sustainment of software-intensive weapon systems. Some SMC program offices have echoed this concern. Over the past decade, virtually all Air Force level guidance for software acquisition and sustainment has been rescinded, and with the recent cancellation of the DoD 5000 series (replaced by DoD 5000.1 and DoD 5000.2 on 12 May 2003), only minimal related guidance remains at that level. This comes as the software contribution to overall weapon system functionality rises to unprecedented levels, and most all software intensive system development efforts are challenged to meet established performance, cost, and schedule baselines. At the same time, there are demands for drastic decreases in acquisition cycle times as well as improved credibility as our leadership works to create an environment of maximum flexibility for program managers. Managers are expected to take on more risk as they attempt to meet these objectives, and we are all asked to apply innovation, continuous improvement, and lessons learned in the acquisition process.

Additionally, the National Defense Authorization Act for fiscal year 2003 contains language in Section 804 that places a new emphasis on software acquisition process improvement. The focus of this improvement activity is on software acquisition planning, requirements development and management, project management and oversight, and risk management. Furthermore, the Act requires metrics for performance measurement and process improvement, a process to ensure that software acquisition personnel have appropriate experience and training, and a process to ensure adherence to software acquisition processes and requirements.

This handbook introduces the software acquisition processes and methods that a project officer should know and follow during system acquisition so that risks inherent in a complex software development are minimized. This guide covers the basic information that is needed for the successful acquisition of a software intensive system. In addition this handbook complies with the DoD Software Acquisition Process Improvement requirements as defined in section 804 of the National Defense Authorization Act of FY 2003.

The processes described in this handbook include both critical and recommended processes for use as part of the system acquisition. The processes are recommended for use within all system program offices (SPOs) that comprise the SMC acquisition organizations. SMC/AXE will be responsible for deploying and maintaining these processes. SMC/AXE will establish methods to promote the employment of the processes and also to ensure they are improved as necessary. The Space and Missile Systems Center (SMC), System Engineering Division developed this document.

Recommendations for changes or improvements to this handbook should be forwarded to Authors: Mr. Michael Zambrana (**Michael.Zambrana@losangeles.af.mil**), Mr. Dennis Singer/D. Singer Engineering, Inc **(dennis@dsingerengr.com**).

Table of Contents

Tables

Figures

1. Introduction

The scope and objective of this Handbook is to provide a Space and Missile Systems (SMC) Software Acquisition (SA) Project Officer with an overview of what is required for successful software intensive program acquisition and management of weapons systems software (SW). In general, the Handbook is to be used as a roadmap and to identify those activities that should be performed but stops short of prescribing specific approaches. This document also identifies the processes that will allow SMC to comply with the Air Force Software Acquisition Process Improvement Strategy. This strategy ensures that SMC complies with all current DoD, OSD, and Section 804 of the National Defense Authorization Act of FY 2003. This Handbook addresses the following:

- Various elements and activities that are recommended during the acquisition of software as part of weapons system.

- Programmatic problems and bottlenecks that typically challenge software acquisitions and the impacts these challenges can have on a program, if not properly managed and controlled.

- The ways to evaluate the software acquisition process so that the Project Officer has the visibility required for effective decision-making.

The processes as depicted in this document allow reasonable flexibility for individual programs to document their software acquisition improvement process in a cost effective manner and also provides guidance on complying with the mandated software acquisition process (section 804) improvement requirements.
Note: See Appendix F for the complete text of the FY 03 Authorization Bill, Section 804.

To effectively review and manage the functions and activities outlined in this Handbook, a certain level of software acquisition and management expertise is required. If you do not have the specific expertise required, use the software expert resources available to you from the Federally Funded Research and Development Center (FFRDC) – Aerospace - and/or System Engineering and Technical Advisor (SETA) contractors.

The issues to be addressed are embodied in these software acquisition processes, which are described in the following sections/paragraphs:

2. Developing Software Acquisition Strategy
3. Ensuring Realistic and Executable Program Baselines
4. Support Request for Proposal (RFP) Preparation
5. Contractor Appraisal
6. Software Development Planning
7. Establishing and Managing Software Requirements
8. Software Documentation
9. Technical Reviews and Audits

10. Integrated Team Management
11. Software Configuration Management
12. Software Quality Assurance
13. Software Test and Evaluation (T&E)
14. Risk Management
15. Metrics
16. Software Support
17. Commercial Off-The-Shelf (COTS)
18. Reuse Software
19. IV&V Contractor Management
20. Training and Experience
21. Software Acquisition Lessons Learned/Best Practices

2. Developing Software Acquisition Strategy

At the beginning of each software intensive system development, the program office must address, as part of the Acquisition Strategy Panel (ASP) process, basic issues that are critical to acquisition program success:

- Software Management Planning: Create a detailed acquisition strategy to meet the acquisition program baseline requirements and fully coordinate and gain approval by the Defense Acquisition Executive (DAE). The SW Integrated Product Team (IPT) will work in conjunction with the user/stakeholder IPT to accomplish the following tasks:
  - Acquisition Strategy (Acquisition Strategy Report, Single Acquisition Management Plan (SAMP), program plan…)
  - Develop Contract Strategy, RFP, Source Selection documents, and Support Source Selection
  - Cost Baseline Development (Ensure realistic and compatible program baselines)
  - Develop SW Portion of Test & Evaluation Master Plan (TEMP)

- Strategic Studies: Conduct studies to support the development of requirements or program decisions needed in the Requirements Analysis (RA) Phase and to identify issues and risks associated with program execution:
  - User Needs Research
  - Industry Capability Assessment / Industry Involvement
  - Assurance (Certification) Process Definition
  - Required Technologies Assessment
  - Ensure use of COTS Avoids Common Pitfalls
  - Identify Areas of High Levels of Reuse and Integration

3. Ensuring Realistic and Executable Program Baselines

Baselines start with a clear set of requirements. To ensure that the software requirements are as fully and clearly described as possible in the early phases of software acquisition, project officers participate in the Request for Proposal (RFP) activity. The task of estimating the required software development effort and schedule is based on as complete as possible an understanding of the requirements and their complexity. This is critical to establishing realistic and executable program baselines.

Programs involving extensive software development often unknowingly sign up to an infeasible schedule or cost baseline. If an estimate of the required software effort and schedule, tied to specific requirements, is not available as the program is formulated, there is a high probability that the early estimate for the overall program cost and schedule will not accommodate the required software development. Likewise, if an estimate is not available when proposals are evaluated in source selection, there is no basis from which to judge the feasibility of effort and schedule proposed by the offerors. Software estimates should be based on the contractor's previous experience in developing and/or integrating software products of similar size, complexity, and scope in a similar domain. See the Cost Management checklist in Appendix C. This Checklist provides additional guidance in estimating the required software development effort and schedule.

3.1 Software Estimates

It is important to remember that software estimates of effort, time, and costs are "living" estimates based on the best information available at the time of the estimate and represents only a "snapshot" based on what is known about the requirements at that point in time. The initial and all subsequent software estimates should be updated, utilizing a standardized process, based on changes to the program requirements, interfaces and other factors, throughout the life of the program.

Computer tools are used extensively to assist in cost estimation. These range from spreadsheets and project management software to specialized simulation and estimating tools. Computer tools reduce the incidence of calculation errors, speed up the estimation process, and allow consideration of multiple costing alternatives. One of the more widely used computer tools for estimating software development costs is the Constructive Cost Model (COCOMO). However, please note that most computer tools for developing estimates for software development use either lines of code or function points as input data. If the number of lines of code or function points cannot be accurately estimated, the output of the tools will not be accurate. The best use of tools is to derive ranges of estimates and gain an understanding of the sensitivities of those ranges to changes in various input parameters. The outputs of the estimating process include the project cost estimates, along with the details used to derive those estimates. The details usually define the tasks by references to the Work Breakdown Structure (WBS). They also include a description of how the cost was derived, any assumptions made, and a range for estimate (e.g. $20,000 +/- $2000.) Another output of the estimating process is the Cost Management Plan. This plan describes how cost

variances will be managed, and may be formal or informal. The following information may be considered for inclusion in the plan:

- Cost and cost-related data to be collected and analyzed.
- Frequency of data collection and analysis.
- Sources of cost-related data.
- Methods of analysis.
- Individuals and organizations involved in the process, along with their responsibilities and duties.
- Limits of acceptable variance between actual costs and the baseline.
- The authority and interaction of the cost control process with the change control process.
- Procedures and responsibilities for dealing with unacceptable cost variances.

Note: For further guidance on software estimating, see Appendix G for a detailed example of the software development estimating process.

A list of widely used software estimating tools include:

- Constructive Cost Model (COCOMO) II
- PRICE-S
- SEER-SEM
- SLIM
- SAGE

3.1.1 Estimates at various points in a program

Estimates can be developed or modified at various points in a program, some of which are:

- During program formulation when initial performance requirements, program cost and schedule estimates, and program strategy are being developed, initial software development size, effort and schedule estimates can be developed.

- During source selection when estimates of proposed development schedules and associated effort could be refined, discrepancies can be resolved to ensure the proposed software development effort and schedule accommodates the required performance and disciplined software development process, and is compatible with the program cost and schedule baselines.

- During the execution of the program when changes drive the need for revised estimates or in conjunction with special reviews or Independent Review Teams (IRTs) or normal reviews, such as Milestone B which usually is established after segment or software requirements have completed their critical design review (CDR).

3.1.2 Cost as an Independent Variable (CAIV)

Evaluation of the cost of proposed requirements vs. operational effectiveness to achieve affordability should be undertaken to ensure that the Government is getting best value. The acquisition strategy should define appropriate cost objectives for acquiring, operating, and maintaining the software systems to be acquired. Cost-performance trades in search of the "best value" solutions should be performed within the constraints of the software performance requirements and affordability.

3.1.3 Cost Analysis and Requirements Description (CARD)

CARD may be required, however, in all cases, the software requirements and cost information adequate for the CCaR (Comprehensive Cost and Requirement) System is required. The software estimates should include the contractor and Government effort, research, training, data, facilities, and reviews, audits and test support costs. This combination of costs reflects the true software Life Cycle Cost (LCC).

4. Support Request for Proposal (RFP) Preparation

The purpose of this activity is to ensure the Request for Proposal (RFP) contains appropriate software acquisition content. Appropriate RFP content should result in proposals that allow for an adequate government review of the offeror's' approach to satisfying the system performance requirements as well as adequate evaluation of the offeror's' development processes.

4.1 RFP Content

Content for acquisition of systems software should be included in section L (instructions to offerors), Section M (software acquisition evaluation standards), Contract Data Requirement List (CDRLs), and Statement of Work (SOW)/Statement of Objectives (SOO) of the RFP, such as:

- Top level mature software development process objectives in the SOW/SOO

- For the acquisition of mission critical and support software, **Section L** of the RFP should require submittal of a draft Software Development Plan (SDP) that defines the offeror's proposed software development processes to be applied during the program life cycle.

- Assure software processes defined by the offeror are reflected in the draft Software Development Plan (SDP), and in the submitted Integrated Management Plan (IMP) and Integrated Master Schedule (IMS), and that they include processes for handling Non-Developmental Items (NDI) software, software as Government Furnished Materials (GFM), and Commercial-off-the-shelf (COTS) software.

- Assure software size and complexity estimates and the resultant software development effort and schedule are consistent with the overall program development schedule and that dependencies between software and firmware elements and hardware elements are identified and maintained. If not, identify the failure to have these dependencies identified as risk areas.

- The technical definition of the computer software architecture and data metamodel, estimated sizing, throughput timing, and growth migration strategy also need to be defined as criteria in **Section L** of the RFP and in the offeror's proposal.

- For acquisition of mission critical software **Section M** of the RFP should define minimum software technical performance requirements criteria for evaluating the offeror's proposal.

5. Contractor Appraisal

The purpose of this activity is to ensure that **Section L** (Instructions to Offerors) of the Request for Proposal (RFP) contains appropriate instructions that allow for an adequate government review of the offeror's approach to satisfying the system performance requirements as well as adequate evaluation of the offeror's' software development processes and planned development approach.

Currently, the SEI Capability Maturity Model – Integrated (CMMI) model and methodology can serve as a representative set of mature software development and management processes. Other appraisal models/methodologies may currently be in use on Department of Defense (DoD) programs.

During source selection the Program Office SA Appraisal Team can utilize the CMMI Model and Methodology as the basis for identifying the strengths and weaknesses of the offeror's software development process. The appraisal results can be used in a number of ways, such as:

- Generating a maturity level or capability level for the offeror's software development organization
- Guidance for source selection decision makers
- A means of reducing acquisition risk for software intensive systems.

5.1 Project Officers Activities For Contractor CMMI Appraisal Preparation

The project officer will Chair the SMC CMMI appraisal team or support chair if assigned from outside the SPO (e.g. home office senior technical expert or advisor). In addition, they will participate in the selection and preparation/training of team members, and assist in preparing and/or reviewing the CMMI Plan and Schedule for the appraisal, determining the appropriate scope and size of the appraisal.

- Prepare RFP instructions reflecting the Contractor Appraisal
- Plan for and conduct site visits, if required
- Evaluate, score and integrate results into source selection
- Define required systems and software developer capability balanced against the needs of the development program including criticality of the program

For a more in-depth discussion of the SEI CMMI go to the SEI web site at: http://www.sei.cmu.edu/sei-home.html:

- Capability Maturity Model Integrated Version 1.1 (Continuous and Staged representations)
- Appraisal Requirements for CMMI v1.1 (ARC, V1.1)
- Standard CMMI Method for Process Improvement (SCAMPI) V1.1: method Definition Document

6. Software Development Planning

Each major software product within the program or a segment of the program should have a form of integrated team approach as an Integrated Product Team (IPT). Additionally, a system-wide, overarching software engineering, integration, and test IPT, which could be referred to as the Software Engineering Integration and Test (SWEIT) IPT, may be established. The following types of IPTs may be established to assist in software development planning and management.

6.1 SW Cost/Performance IPT

The Cost/Performance IPT is responsible for integrating and evaluating all system-level, cost performance, trade-off analyses. It recommends performance or engineering and design changes to meet Capability Development Document (CDD) and any other software or requirements threshold values for system-level requirements. Office of the Secretary of Defense (OSD) and the Services may be represented in this IPT.

6.2 SW Cost IPT

The SW Cost IPT may be a multi-service/agency IPT led by the cognizant SMC Program/Project office, depending on the scope and involvement of other services or agencies. It will be established to develop and coordinate the Program Office Estimate (POE) and Service Cost Position. This IPT works under the Cost IPT, headed by the Air Force Cost Analysis Agency.

6.3 SW Acquisition IPT

The SW Acquisition IPT is chaired by the cognizant SMC Program/Project office/officer. This IPT is responsible for developing a cost-effective and responsive acquisition strategy that satisfies all civil and military services requirements being satisfied by software. This IPT will also develop the program Single Acquisition Master Plan (SAMP). Members may include representatives from the user community and other agencies/services, as appropriate to the project.

6.4 SW Test IPT

The SW Test IPT will be chaired by cognizant SMC Program/Project office, with support of the Responsible Test Organizations and other Services stakeholders. The scope of this IPT is to develop and maintain all *Test and Evaluation Master Plans* (TEMPs) and support the RFP development process by ensuring requirements are testable. This IPT is also responsible for addressing the development and application of Modeling and Simulation in testing activities.

7. Establishing and Managing Software Requirements

The Requirements Analysis (RA) Phase will define the software requirements that will be demonstrated to meet specific requirements in the CDD. The purpose of this activity is to ensure that software requirements are defined, complete, verified, consistent, testable and traceable to higher level system and component requirements and lower level (software component) design and implementation. The Project Officer will manage the strategic planning and studies, systems engineering and program integration functions throughout this phase. The Project Office should form a SW IPT composed of government, FFRDC, and SETA support to accomplish these tasks. See appendix C for the Requirements Engineering checklist.

7.1 Systems Engineering

Ensure that all project software requirements are defined in a systematic way, with traceability to system level operational requirements documents, and any related users Service CDDs. The SW IPT will address the following specific responsibilities:

- Requirements Definition, Analysis, Development, and Management
- Develop Draft Interface Control Documents/Drawings (ICDs) (Non-Proprietary and Hardware Independent)
- Software Security modules and Program Interface and Initial Review

See Appendix A Lessons Learned for greater coverage.

8. Software Documentation

Documentation is essential to accomplish software development, maintenance, and management effectively and efficiently. For a major portion of the software development activity, documents are the only visible and permanent products. Without software documentation, it may be very expensive or impossible to support the software after system deployment. Evaluation of accomplishments and control of activities is only possible when those accomplishments are documented. In some cases when insufficient, poor or no documentation was provided, it was necessary to "reverse engineer" the software or abandon the software to start all over again, a very expensive and time consuming process.

8.1 Purpose and Value of Documentation

Software documentation supports the basic requirement to provide a well defined and consistent software baseline during the acquisition life cycle that can resolve the turnover of personnel, provide management visibility, user visibility, and provide the requirements and design information to the developers.

8.1.1 Turnover of Personnel

Virtually every computer system has some turnover of personnel during development and to deployment. Because turnover is anticipated, the computer software must be written so that it will be easily understood by new personnel. Documentation used as an introductory and quick reference tool provides a valuable aid for new project personnel who are not intimately familiar with the computer software. Documentation used during software maintenance must be kept up to date. If the documentation is not current, then it will impede the maintenance effort and result in a useless expenditure of time, effort, and money.

8.1.2 Government Visibility (Insight and Involvement)

Maintaining technical insight and resolving development issues are prime responsibilities of the program office. Government visibility into the development process is necessary to allow control and provide redirection if necessary. This includes day-to-day oversight into the acquisition process, effective interface with the contractor, definition and monitoring of metrics and TPMs, and review of selected software development documentation in order to ensure the delivered software products satisfy technical, cost, schedule, and supportability.

Software documentation (i.e., specifications, plans, and reports) and software reviews provide the project office with evidence of the project's progress. The software should be written, reviewed, and tested in a modular and incremental manner so that the developer can see intermediate results. For example, the user interface software might be written and tested first to gain visibility into this sensitive area of development. Few project office’s personnel have the time or the required technical knowledge to become intimately familiar with everything about the system software they control. Good documentation, frequent reviews, and modular development will provide them with sufficient visibility into what the program does and a measure of the progress achieved

in completing the development. With this knowledge, both the Project Officer (PO) and contractor's managers can better perform their controlling functions.

8.1.3 User Visibility

User visibility into the development process is necessary to ensure that the completed system meets requirements and is " user friendly." Visibility to the users should occur as soon as possible in the development process to ensure that the users' requirements can be built into the system rather than added later, at much greater expense. To achieve this early user input, it is necessary for the software development contractor and PO to inform the users of the software's capabilities. For example, the users should have the opportunity to review the requirements specifications before the software is designed, and to review the design documents (containing such items as display formats) before the design is coded. If the user documentation is not written until the program is already complete, then user requirements may have to be added, possibly causing massive changes, schedule slippage, and significant additional cost.

8.2 Software Documentation Categories

There are two general categories of software documents: the plans that describe how the software development and maintenance effort will be carried out, and the specifications that describe the software product that is being developed, tested, and maintained. Each of these document types is discussed in the following sections.

8.2.1 Software Planning and Maintenance Documents

These documents describe the plans for the software being developed and maintained. Two such plans are critical to the software development and maintenance process: the Software Development Plan (SDP), and the Software Maintenance Plan.

8.2.1.1 Software Development Plan

The SDP is the guide that controls all of the software developer's activities and is used to provide the government insight into the contractor’s organization responsible for developing the software. This plan should be tailored specifically for the particular project, but should include proven processes, methods and procedures from past successful software development projects. It is written and carried out by the software development contractor and is the vehicle by which the development contractor tells the Project Officer how the contractor will do his job. Because of the importance of the Software Development Plan, it should be thoroughly reviewed by the Project Officer (or by designated personnel) before acceptance and approval. A draft of the SDP should be specified in the Request For Proposal (RFP) as a compliant document and included as part of the contractor’s proposal. The Software Development Plan must cover the following topics:

- Organization - Who is responsible for each software development task (e.g., design, code, test, etc.) and what is the reporting chain of people and organizational groups?
- Management and Technical Controls - How will the software development be managed and what management controls will be employed?

- Schedule and Milestones – What are the detailed schedule and specific milestones for the software development effort, and how do they relate to the overall system development schedule?
- Status Monitoring - How will management know where the project is with regard to the schedules?
- Documentation - What documents will be produced and when? What formats will be employed and what automated facilities will be used? How will the documents be reviewed and approved?
- Standards, Practices, and Guidelines - What specific internal standards, practices, and guidelines will be followed in the design, code, and test activities?
  - How will this policy be enforced and how will required deviations be approved?
- Development and Test Resources - What support software and hardware is required and how will this software and hardware be obtained, maintained, and documented? Which of this software and hardware is deliverable to the acquisition agency, and how and when will it be delivered?
- Software Quality Assurance – What methods will be used for ensuring the integrity and quality of all software processes and products (e.g., reviews and walkthroughs, structured testing, automated analysis, etc.)?
- Error Reporting – How will errors in software products be documented?
  - What accountability approaches will be used to make certain that all detected errors are corrected?
- Configuration Management - What software products will go under configuration control and when?
  - What configuration control boards will be established and who will make up these boards?
  - Will there be internal change control before formal Government change control?
  - How will software configuration management interact with other system configuration management activities?
- Security - How will classified data and software products be controlled?
  - How will hardware facilities be installed, controlled, and operated to enable the processing of classified data?

NOTE: For detailed suggested content of the Software Development Plan see Data Item Description (DID) DI-MCCR-80030A in appendix E.

8.2.1.2 Software Maintenance Plan

The Software Maintenance Plan (also known as the Computer Resources Lifecycle Management Plan--CRLCMP) should be prepared by the agency that is acquiring and will be responsible for maintaining and using the software. The project officer should make certain that a team of DoD personnel from the acquisition, maintenance or logistics, and user organizations is assembled for the plan preparation. This team, usually called the Computer Resources Working Group (CRWG), should be assembled very early in the development effort, even before the software development contract is awarded, if possible.

The first major decision of the CRWG and a key element of the Software Maintenance Plan is the selection of the organization that will be responsible for the software maintenance after delivery of the system. The organization could be either a contractor or a government support organization. The latter is called organic maintenance.
The major topics covered in the Software Maintenance Plan are:

- Organization – Who will be responsible for software maintenance throughout the system's life? Who are the focal points in the acquisition and using organizations that work with the maintenance organization?
- Transfer of Responsibility – What conditions must be established before responsibility for software integrity passes from the acquisition to the maintenance organization? How and when will transfer of responsibility take place?
- Documentation – What documentation is needed to enable life cycle support? How and when will it be obtained? How will its integrity be evaluated?
- Support Software Needs – What support software (e.g., compilers, simulators, etc.) is needed for life cycle support? How and when will it be obtained? How will its correctness be established and how will it be controlled?
- Equipment Needs – What computers and other hardware are needed? How and when will these computers and hardware be acquired and maintained?
- Personnel Needs – How many people are needed to staff the support facility? When are they needed? What skills are needed and how will these skills be obtained?
- Configuration Management – What actions will be taken to establish the initial software's baseline configuration and to maintain the software's integrity throughout the system's life cycle?
- Funding - How much will everything cost and who will pay for it? How much money is needed each quarter or fiscal year?

See appendix C for the Sustainment checklist.

8.2.2 Software Development Documents

The suite of software development documents consists of the core development documents and essential support documents. Each of these document types is discussed in the following sections.

8.2.2.1 Software Development Core Documents

For the software development contractor the following Core documents are essential for the development of each Computer Software Configuration Item (CSCI). Table 8.2.2.1-1 "Core Software Development Documents" describes each document and it's purpose.

Table 8.2.2.1 –1 Core software development documents

Document Type	Document Purpose
Software Requirements Specification (SRS)	What the software must do.
Software Design Document (SDD)	How the software will do it.
Software Test Plan (STP)	How it will be shown that the software fulfills its requirements.
Software Test Description (STD)	The specific inputs, scenario, and acceptance criteria for each test.
Software Test Report (STR)	The results obtained when the test procedures were conducted.
Software Product Specification (SPS)	What was coded, tested, and delivered.
Version Description Document (VDD)	What is included in a particular version of the software that is delivered?

Note: All of documents listed in Table 8.2.2.1-1 are required for each CSCI in every software development activity. The Software Requirements Specification and the Software Design Document, after approval by the Project Officer (PO), will serve as baselines for subsequent software development activities. The Software Test Plan, Software Test Description, and Software Test Report also should require approval by the PO, but they do not become baselines since they do not define mandatory characteristics of the final software product.

8.2.2.2 Other Essential Support Documents

The System Specification is not strictly a software document, but must precede and control the Software Requirements Specification. The Interface Requirements Specification is important for large CSCIs. Where there is multiple interfacing CSCIs, it acts as a linkage point between separate Software Requirements Specifications and Software Design Documents. The Software Users Manual (SUM) is essential for those applications in which a human operator has direct access to and control over software operation. In such applications, the SUM is an expansion of, and equal in importance to, the Software Requirements Specification. There are applications in which the user may not be aware of the software or directly control the software; in these cases, a SUM may have no meaning and may be omitted.

8.2.2.3 Document Traceability

These documents do not stand by themselves, but must maintain bi-directional traceability between each other. Each software requirement in the Software Requirements Specification must link or trace back to one or more requirement in the System/Segment Specification. Similarly, each function or module in the Software Design Document must trace to one or more requirements in the Software Requirements Specification. Each test defined in the Software Test Procedures must trace through the code and design segments that are tested back to the pertinent

requirements in the Software Requirements Specification and System/Segment Specification. There should be no breaks in this requirements- design-code-test traceability.

8.3 Documentation Review

The quantity of documentation prepared for a software development effort is large and can severely tax the ability of the Project Officer (PO) to perform timely and effective reviews. Some level of document review by the PO is essential if only to motivate the software development contractor to prepare complete and correct documentation. Only trouble can result from the attitude that "it doesn't matter what I write because nobody reads it anyway." Given limited PO resources to review the documentation, there are some areas that are the most productive to examine because they are usually the ones that are the most neglected and troublesome. The points that should be especially closely examined are:

- Software Planning (i.e., Software Development Plan)
- Software Sustainment Planning (i.e., CRLCMP)
- Software Risk Management Planning
- Software Metrics
- Software Requirements Specification (SRS) also traceability of S/W Requirements to System Specification
- Software Test Plans and Procedures
- Software Configuration Management Plan
- Software Quality Assurance Plan

Other documents besides those mentioned above are important in a software development and maintenance effort. Table 8.3-1 "Essential Documents for Software Development" describes each document and it's purpose.

Table 8.3 –1 Essential Documents for Software Development

Document Type	Document Purpose
System or System Segment Specification (SSS)	What the total system must do.
Interface Requirements Specification (IRS)	What requirements are imposed on the software-to-software and hardware-to-software interfaces.
Interface Design Document (IDD)	How the CSCIs communicate with each other and with the hardware.
Software Users Manual (SUM)	How an operator uses the software and interprets its results.

8.4 Contract Data Requirements List (CDRL) Selection

The number and types of required documents should to be kept to the minimum that will permit effective government insight and oversight. Utilization of processes, practices, and documents, as formatted by the contractor shall be accepted as much as possible. The following list is the minimally required processes and documentation that the contractor should make available to the Government, if required with regard to Software.

- Software Configuration Management Plan (SCMP)
- Software Documentation Plan
- Software Design Document (SDD)
- Software Integration Test Plan
- Software Product Specification (SPS)
- Software Quality Assurance Plan/Process (SQAP)
- Software Requirements Specification (SRS)
- Software Test Description (STD)
- Software Test Plan (STP)
- Software Test Report (STR)
- Version Description Document (VDD)

9. Technical Reviews and Audits

As important as documentation is to the successful management of a software acquisition effort, there is no substitute for face-to-face meetings between the contractor's development team and knowledgeable representatives of pertinent Program Office technical, management and user organizations. These meetings are both formal and informal; they are not a substitute for adequate documentation, but rather should provide insights into the content and intent of that documentation.

The formal meetings are called reviews and audits, as they are described in detail in Mil-Std-1521B (Technical Reviews and Audits for Systems, Equipments, and Computer Programs). These reviews and audits are conducted for hardware and software. It is a management decision whether to hold combined hardware and software reviews and audits, or to hold separate sessions. If the latter course is chosen, then a system-level review should be held at the conclusion of the separate hardware and software reviews. Informal reviews, also known as Technical Interchange Meetings (TIMs), should be held between formal reviews as defined in Mil-Std-1521B. An active Program Office role is mandatory in the formal reviews and audits process. See appendix C for the Formal Reviews checklist.

The scheduling of formal reviews and audits is directly tied to the contractual obligations of the contractual obligations of the development organization (contractor). The specific details of formal reviews and audits are outlined in the following paragraphs and are contrasted with informal reviews in table 9.1-1.

9.1 Formal Reviews and Audits

There are seven formal reviews and audits in a software development activity as defined in Mil-Std-1521B. These reviews and audits are very important milestones in the software development effort. They should receive major attention both from the Project Officer and from the development contractor. At the end of these formal reviews the Project Officer must decide whether or not to approve the products being reviewed and to permit the software development contractor to continue its efforts along the paths presented in the review. If the products of the development contractor contain significant deficiencies that make them unsuitable for the software development phase, then the Project Officer must determine this fact then give appropriate technical and management direction to the software development contractor. A major reason for cost and schedule overruns is the expense and time required to fix requirements and design errors that are discovered late in the test phases.

Table 9.1 - 1 Characteristics of Formal and Informal Reviews

	FORMAL REVIEWS	**INFORMAL REVIEWS**
FREQUENCY	At key decision points and the end of each software development phase.	Throughout the development effort, during each phase.
REVIEW	Interim products of software	Preliminary version of a

DATA	development phase.	small segment of product.
PRIMARY PURPOSE	Obtain approval to begin next phase.	Find errors in product segment.
REVIEWERS	• SW Acquisition Project Officer • Functional Experts • System Users • Maintenance Organization • Quality Assurance • V & V Team (if one exists for the program)	• Peers of product's producer • Next phase user of product • Previous phase developer
REVIEW LENGTH	One day to several weeks.	Less than 90 minutes
REVIEW OUTCOME	• Technical direction • Review minutes • Action items • Approval, contingent approval, or disapproval of reviewed documentation	Issues for resolution

The reviews and their place in the software development cycle are as follows (see appendix C for the review checklist`):

Table 9.1 - 2 Sequential Orders of Formal and Informal Reviews

Review	**Description**
Software Specification Review (SSR):	This review takes place after software requirements have been defined.
Preliminary Design Review (PDR):	This review takes place after the software architecture has been determined and before detailed design is begun.
Critical Design Review (CDR):	Review takes place after detailed design is complete and before coding begins.
Test Readiness Review (TRR):	Before system testing has begun.
Functional Configuration Audit (FCA):	Review takes place after software has been coded and tested.
Formal Qualification Review (FQR):	Review takes place immediately before software delivery.
Physical Configuration Audit (PCA):	After software has been coded, tested and integrated on actual system hardware.

9.2 Conditions for Successful Formal Reviews/Audits

For formal reviews or audits to be successful, these conditions are described in the following subsections:

9.2.1 Timing

The review or audit should be held only when the software development contractor has completed the documents to be reviewed and is awaiting approval of documentation prior to beginning the next development phase.

9.2.2 Preparation

Prior to the review or audit, the development contractor must deliver the documents to be reviewed (Data Package) to the Program Office for preliminary review.

9.2.3 Participants

Government representatives at the review or audit should be by people who are familiar with the documentation to be reviewed (reviewed the data package) and have been involved in the preceding reviews or audits for the program.

9.3 Additional Guidance for Formal Reviews and Audits

Additional guidance for formal reviews and audits are:

- Documentation that is to be reviewed must be delivered to the Project Officer sufficiently in advance of the review to permit the review team to perform a preliminary review of the content.

- The Project Officer should ensure that designated reviewers are the best qualified to perform the review. Representatives from the following organizations should participate in the formal review:

    - Software acquisition personnel from the Program Office
    - Personnel from user organizations
    - Personnel from the organization that will be responsible for software maintenance and lifecycle support
    - Quality assurance personnel from the Program Office or related support organizations
    - FFRDC/SETA support

- The Project Officer should hold a pre-review meeting the week before the formal review. The meeting should outline goals of the formal review, known deficiencies in the review data package, any anticipated problems in the contractor's presentation and the government's position for these problems.

10. Integrated Team Management

An integrated team (also known as an "Integrated Product Team" or IPT) is composed of relevant stakeholders who generate and implement decisions for the work product being developed. The members of the integrated team are collectively responsible for delivering the work product. The integrated team receives its assignment from its sponsor. The sponsor of an integrated team is a person or a group (e.g., project manager or even another integrated team) who can assign work tasks and provide resources. Team members may include customers, suppliers, and other stakeholders outside of the organization as appropriate to the product being developed.

One should make sure that integrated team members:

- Provide the needed skills and expertise to accomplish the team's tasks
- Provide the advocacy and representation necessary to address all essential phases of the product's life cycle
- Collaborate internally and externally with other teams and relevant stakeholders as appropriate
- Share a common understanding of the team's tasks and objectives
- Conduct themselves in accordance with established operating principles and ground rules
- Have clearly defined and commonly understood objectives, tasks, responsibilities, authority, and background (of vertical and horizontal interfaces) in order to provide a strong basis for implementing the integrated team.

See Appendix A Lessons Learned for greater coverage.

11. Software Configuration Management (SCM)

In software development and other projects, CM is the process of controlling and documenting change to a developing system. It is part of the overall change management approach. As the size of an effort increases, so does the necessity of implementing effective CM. It allows large teams to work together in a stable environment, while still providing the flexibility required for creative work. CM in a software environment is an absolute necessity. See appendix C for the SCM checklist. CM in general has three major purposes:

- Identify the configuration of the product at various points in time.
- Systematically control changes to the configuration.
- Maintain the integrity and traceability of the configuration throughout the product life cycle.

CM accomplishes these purposes by answering and recording the answers to:

- Who makes changes?
- What changes are made?
- When are changes made?
- Why are changes made?

Effective CM provides the following essential benefits to a project:

- Reduces confusion and establishes order.
- Organizes the activities necessary to maintain product integrity.
- Ensures correct product configurations.
- Limits legal liability by providing a record of actions.
- Reduces lifecycle costs.
- Enables consistent conformance with requirements.
- Provides a stable working environment.
- Enhances compliance with standards.
- Enhances status accounting.

11.1 SCM Process Description

The SCM process is a subset to the program overall Configuration Management process. The process presented in this section is applicable to Software Configuration Management (SCM).

11.1.1 Functions of Software Configuration Management

Software Configuration Management is comprised of four primary functions:

- Identification
- Control
- Status Accounting
- Auditing

11.1.1.1 Identification

This function identifies those items whose configuration needs to be controlled, usually consisting of software, and software documentation. These items would probably include such things as specifications, designs, data, documents, drawings, software code and executables, components of the software engineering environment (compilers, linkers, loaders, hardware environment, etc.), project plans and guiding documents, especially the project requirements. A schema of names and numbers is developed for accurately identifying products and their configuration or version level. This must be done in accordance with project identification requirements. Finally, a baseline configuration is established for all configuration items and systems. Any changes to the baselines must be with the concurrence of the configuration control organization.

11.1.1.1.1 Selection of Software Configuration Items

Selection of software configuration items is accomplished by review and analysis in accordance with:

- Contract Statement of Work
- Software Compliance Standard e.g., Mil-Std 498
- Configuration Management Compliance Standard e.g., MIL-STD-973

The following program product teams or activities conduct selection of software configuration items:

- Systems Engineering Team
- Software Engineering Team
- Configuration Management
- Quality Assurance

11.1.1.1.2 Developmental Software Configuration Identification

Developmental software configuration identification encompasses four phases of the development cycle:

- Preliminary design
- Critical design
- Software Test
- System integration and test

SCM is initially responsible for verifying the establishment and authentication of the allocated baseline, also known as the software requirements specification or B-5. Once SCM has verified the authentication of allocated baseline(s), establishment of the developmental configuration and preliminary design can begin.

The Software Development Library (SDL) as specified in the Software Development Plan (SDP) manages the developmental configuration of each CSCI. The software engineering environment and software test environment tools (to verify released builds) must be released to the SDL. The SDL places released software in a control database. This controlled database is used to manage all changes to released software from the initial release through the remainder of the software life cycle. The control database is described in the SDP.

11.1.1.2 Control

Software Configuration control establishes procedures for proposing or requesting changes, evaluating those changes for desirability, obtaining authorization for changes, publishing and tracking changes, and implementing changes. This function also identifies those persons and organizations that have authority to make changes at various levels (configuration item, assembly, system, project, etc.,) and those who make up the configuration control board(s). Additionally, various change criteria are defined as guidelines for the control organizations. Different types of configuration items or different systems will probably need different control procedures and involve different people. For example, software configuration control has different needs and involves different people than communications configuration control and would probably require different control rules and a different control board.

11.1.1.2.1 Software Configuration Control Board (SCCB)

The SCCB is established after completion of the Architectural Design Review (ADR). The CCB is responsible for change coordination with the program office and supervision of the Software Change Control Board (SCCB). The CCB is cognizant over the functional baseline documents, the hardware allocated and product baseline documents, and the software allocated baseline documents after establishment. The SCCB is responsible for coordination of software problem reporting, the maintenance of the software allocated and product baselines, software development library management, Non-Development Software (NDS) change management, and software release authority.

11.1.1.3 Status Accounting

Status accounting is the documentation function of SCM. Its primary purpose is to maintain formal records of established configurations and make regular reports of configuration status. These records should accurately describe the product, and are used to verify the configuration of the system for testing, delivery, and other activities. Status accounting also maintains a history of change requests and authorizations, along with status of all approved changes.

11.1.1.4 Auditing

Effective SCM requires regular evaluation of the configuration. This is done through the auditing function, where the physical and functional configurations are compared to the documented configuration. The purpose of auditing is to maintain the integrity of the baseline and release configurations for all controlled products. Auditing is accomplished via both informal monitoring and formal reviews.

11.2 SCM Process

Understanding of what SCM is supposed to accomplish is one thing. As with most project activities SCM begins with planning. With a plan, configuration baselines can be established. Following this initial configuration identification, the cyclical configuration control process is put into motion. These three major CM implementation activities are shown in Figure 11.2-1.

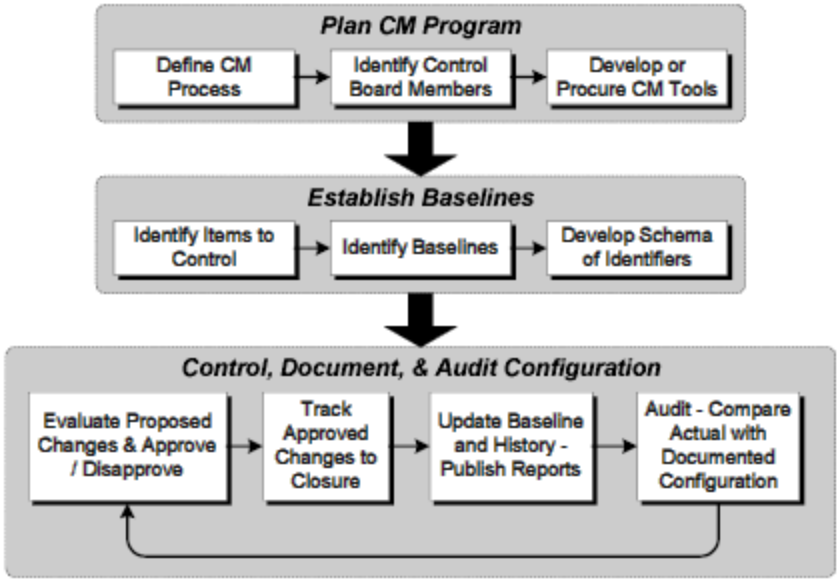


Figure 11.2-1 Software Configuration Management Implementation Process

11.2.1 Planning

Planning begins by defining the SCM process and establishing procedures for controlling and documenting change. A crucial action is the designation of members of the Software Configuration Change Board (SCCB). Members should be chosen who are directly or indirectly involved or affected by changes in configuration. For example, a software configuration change board would be populated with representatives from different software teams, but software affects many more aspects of a project. There should also be representatives from the hardware, test, systems, security, and quality groups, as well as representatives from project management and possibly other organizations. Various software tools exist which can facilitate the SCM process flow and maintain configuration history. Use of a SCM software tool is highly recommended. The temptation will be to choose a tool because it looks good in a demonstration and then build the SCM process around it. The process should be defined first, and then a tool chosen to facilitate the process. Software configuration management planning

should be documented in the Software Configuration Management Plan (SCMP). The SCMP must contain at a minimum the following information:

- Introduction. Overview of the SCM process, the purpose and scope of the SCMP.
- Management. Procedures for establishing SCM organization, and for assigning tasks/responsibilities to all units in the organization.
- SCM Activities. Includes configuration identification, configuration control, configuration status accounting and report, and audits/reviews. Include in the description how these activities are to be implemented, which units are responsible for which activities, etc.
- Tools, Techniques, and Methodologies. Tools and methodologies to be used for implementing SCM activities.
- Records Collection and Retention. Procedures for identifying, assembling, filing, storing, maintaining, and disposing of SCM documentation.
- SCMP Maintenance. How the SCMP will be reviewed, updated, or revised while in use.

11.2.2 Establishing Baselines

Once the SCM program exists on paper, it must be determined just what configurations it will control. The second major step of implementing effective SCM is identifying what items, assemblies, code, data, documents; systems, etc. will fall under configuration control. With the configuration items identified, the baseline configuration must be identified for each item. For items that already exist it may prove to be nothing more than examining or reviewing, and then documenting. For those items that have not been developed yet, their configuration exists in the requirements database or in the project plans. Until they come into physical or software reality, changes to their configuration will consist only of changes to the requirements or plans. Another essential activity in this step is developing a schema of numbers, letters, words, etc. to accurately describe the configuration revision, or version, for each general type of configuration item. There may be project requirements which dictate some type of nomenclature, or there may be an organizational or industry standard that can be used as the basis for configuration identification.

Baselines are composed of all CIs describing and pertaining to a system at a point in time. The Configuration Manager uses baselines to maintain traceability of the changes in the CI throughout the life cycle.

The items included in system baselines are stored and maintained in an electronic media format once changes are made. A current baseline consists of the previous baseline plus all approved changes to that baseline. Figure 11.2.2-1 depicts the composition of the baselines including software and related documents.

Functional Baseline	Operation Concept Description (OCD)
	System/Subsystem Specification (SSS)
	Software Development Plan (SDP)
	Software Configuration Management Plan (SCMP)
	Software Quality Assurance Plan (SQAP)
Allocated Baseline	Functional Baseline plus approved changes
	System/Subsystem Design Description (SSDD)
	Software Requirements Specification (SRS)
	Interface Requirements Specification (IRS)
Development Configuration	Allocated Baseline plus approved changes
	Software Design Description (SDD)
	Interface Design Description (IDD)
	Database Design Description (DBDD)
	Software Test Plan (STP)
	Software Test Description (STD)
	Software Test Report (STR)
	CSCI Source Code/Executable Code
	CSCI Database
	CI Non-developmental Items
Product Baseline	Developmental Configuration plus approved changes
	Product Specification (PS)
	Installation Plan (IP)
	Transition Plan (TrP)
	Version Description (VD)
	User Manual (UM)
	Input/Output Manual (IOM)
	Center Operator Manual (COM)
	Computer Operation Manual (COM)
	Computer Programming Manual (CPM)
	Firmware Support Manual (FSM)
	CSCI Source Code/Executable Code
	CSCI Database
	CI Non-developmental Items

Figure 11.2.2-1 Configuration Identification of Baselines

11.2.2.1 Functional Baseline

The internal software allocated baseline is established after PDR with the SCCB functioning as the change control authority over the software requirements specifications. The functional baseline is established following the completion of the System Design Review. The authenticated functional baseline consists of System Specification CDRL.

11.2.2.2 Allocated Baseline

The allocated baseline is established following the CDR for hardware and TRR for software.
The authenticated allocated baseline includes the following CDRLs:

- Prime Item Development Specification (B1)
- Software Requirements Specification (SRS)
- Interface Requirements Specification (IRS)

11.2.2.3 Development Configuration

Development Configuration identification encompasses four phases of the development cycle: preliminary design, critical design, software test, and system integration and test. The Developmental configuration is not considered a baseline, but is viewed as a progression from design allocations to an auditable product design. SCM is initially responsible for verifying the establishment and authentication of project-specific allocated baselines to ensure the stability of the software development effort.

11.2.2.4 Product Baseline

The Product Baseline is established when the development configuration is deemed ready for Functional Configuration Audit (FCA) and Physical Configuration Audit (PCA). Completion of the FCA/PCA will require the following items be released, delivered, and approved by the program office:

- Identified changes to the prior baselines,
- System/Segment Design Document
- Processing Software Product Specification
- Version Description Document
- Prime Item Product Specification
- Software Design Document for each CSCI
- Product Drawings and Associated Lists
- Commercial Drawings and Associated Lists
- Software Users Manual
- Firmware Support Manual
- Interface Design Document
- Interface Control Document
- Subsystem Interface Control Document
- Interface Control Document
- Computer Resources Integrated Support Document

11.2.3 Controlling, Documenting, and Auditing

When the baselines have been established, the challenge becomes one of keeping the actual and documented configurations identical. Additionally, these baselines must conform to the configuration specified in the project requirements. This is an iterative process consisting of the four steps shown in Figure 11.2-1. All changes to the configuration are reviewed and evaluated by the appropriate configuration control representatives specified in the SCM plan. The change is either approved or disapproved. Both approvals and disapprovals are documented in the SCM history.

Approved changes are published and tracked or monitored until they are implemented. The appropriate configuration baseline is then updated, along with all other applicable documents, and reports are published and sent to affected organizations indicating the changes that have occurred. At selected time intervals and whenever there appears to be a need, products and records are audited to ensure that:

- The actual configuration matches the documented configuration.
- The configuration is in conformance with project requirements.
- Records of all change activity are complete and up-to-date.

The controlling, documenting, auditing cycle is repeated throughout the project until its completion.

11.2.4 Updating the SCM Process

It is unlikely a perfect SCM program will be assembled during the initial planning stage. There will be learning and changes in the program that indicate a need for adjustments in the SCM process. These may be any mixture of modifications to make it more efficient, responsive, or accurate. When changes in the SCM process are needed, consider them as you would any other changes, get the approval of all participating organizations, and implement them as appropriate. It would be ironic indeed to have an unchanging change process.

12. Software Quality Assurance

Software Quality Assurance (SQA) is a group of related activities employed throughout the software life cycle to positively influence and quantify the quality of the delivered software. SQA is not exclusively associated with any major software development activity, but spans the entire software life cycle. It consists of both process and product assurance. Its methods include assessment activities such as ISO 9000 and CBA IPI (CMM-Based Appraisal for Internal Process Improvement), analysis functions such as reliability prediction and causal analysis, and direct application of defect detection methods such as formal inspection and testing.

The focus of SQA is to monitor continuously throughout the software development life cycle to ensure the quality of the delivered product. This requires monitoring both the processes and the products. In process assurance, SQA provides management with objective feedback regarding compliance to approved plans, procedures, standards, and analyses. Product assurance activities focus on the changing level of product quality within each phase of the life cycle, such as the requirements, design, code, and test plan. The objective is to identify and eliminate defects throughout the life cycle as early as possible, thus reducing test and maintenance costs. See appendix C for the Formal Reviews checklist and appendix D for the software quality assurance attributes list.

Examples of software items that undergo software quality assurance throughout the software life cycle are:

- Development
  - Requirements (System/Software Requirement Specification)
  - Architecture (System/Software Architecture)
  - Detailed Design (Software Detailed Design)
  - Software code (Ada, C++, FORTRAN, Java, Assembly)
  - Machine code (Object code, Executable code)
  - Testing and Qualification Documents (Test Plan, Tests documentation, Test Reports, Software Qualification Requirements, Software Qualification Test Report)
  - Acceptance testing and report (Acceptance Tests documentation, Acceptance Test Report)
  - Software installation (Software Installation Plan)
  - Software user's manuals
- Maintenance
  - Post-installation data (Problem and Modification Report, Migration Plan, Withdrawal Plan)
- Quality Assurance
  - Quality Assurance (Quality Assurance Plan)

12.1 Process Assurance

Software quality assurance is defined as the set of systematic activities providing the evidence of the ability of the software process to produce a software product that is fit for use. SQA oversight provides management with unbiased feedback on process compliance so process lapses can be addressed in a timely fashion. It provides management with an early warning of risks to product quality and can provide recommendations to address the situation. It is essential that the software quality assurance personnel have a reporting path, which is independent of the management responsible for the activities audited, and the associated daily conflicts generated by schedule and budget. Independent oversight, through various methods, encourages adherence to the official process. Locating an appropriate level of management where SQA will have frequent access, active support, and be above the conflicts of interest may be a difficult but necessary step. The methods typically used to accomplish process assurance include SQA audit and reporting, assessment and statistical process control analysis.

12.2 Product Assurance

Assurance that the product performs as specified is the role of product assurance. This includes “in process,” or embedded, product assurance, as well as some methods that involve independent oversight. The purpose of embedded SQA processes is product quality assurance. The activities are part of the development life cycle which will “build-in” the desired product quality. This focus allows identification and elimination of defects as early in the life cycle as possible, thus reducing maintenance and test costs. Embedded SQA methods include formal inspection, reviews, and testing. Independent oversight functions can also be a part of product assurance. An independent test function, or testing which is witnessed by an independent entity such as SQA, is one method of providing product assurance. Other options include tests witnessed by customers, expert review of test results, or audits of the product.

12.3 Methods and supporting technologies

Many methods are used to perform the process and product assurance functions. Audits are used to examine the conformance of a development process to procedures and of products to standards. Embedded SQA activities, which have the purpose of detecting and removing errors, take a variety of forms, including inspection and testing. Assessment is another method of process assurance. Analysis techniques, such as causal analysis, reliability prediction and statistical process control, help ensure both process and product conformance.

12.3.1 Audit

Auditing is a method used in both process and product assurance. Audits are embedded into the software life cycle, as well as being performed as part of SQA.

An SQA audit is performed to “determine the adherence to established standards and procedures.” Evaluation of the sufficiency or effectiveness of the procedures or standards is occasionally part of an SQA audit. This type of audit examines records, as opposed to products, according to a sampling process to determine if procedures are

being followed correctly. An external auditor who is not part of the software project often performs such an audit.

In contrast, an embedded audit examines products to determine if the software products conform to standards and if project status is accurate. An independent auditor may perform this function or evaluate the records of such an audit that was performed by the development process. For documents, the audit is often performed manually. For code, it may be done manually or by an automated tool.

12.3.2 Embedded SQA Error Detection Methods

The basic SQA Error Detection Methods consist of:

- Formal Inspections
- Reviews
- Walkthroughs
- Testing

12.3.2.1 Formal Inspection

Formal inspection is an examination of the completed product of a particular stage of the development process (such as design or code), typically employing checklists, expert inspectors, and a trained inspection moderator. The objective is to identify defects in the product. There are many techniques of doing inspections, but many follow the methods developed by Michael Fagan over 20 years ago. Certain projects that have an effectively performing inspection process report better than 80% defect detection rates.

12.3.2.2 Reviews

Reviews are also applied as an alternative to formal inspections as an SQA method. Informal design and code review methods are difficult to quantify since they are generally done at the discretion of the product author, do not follow a detailed process and are not reported at the project level. Informal review is a valuable alternative if the more effective formal inspection is not used. The term “review” is also used to refer to project meetings (e.g., a product design review) which emphasize resolving issues and which have a primary objective of assessing the value of the product.

12.3.2.3 Walkthroughs

Walkthroughs are meetings in which the author of the product acts as presenter to proceed through the material in a stepwise manner. The objective is often raising and/or resolving design or implementation issues. Walkthroughs tend to be informal and lacking in “close procedural control.”

12.3.2.4 Testing

Testing is a dynamic analysis technique that has the primary objective of error detection. Testing of software is performed on individual components during intermediate stages of development, subsystems following integration, and entire software systems. It involves execution of the software and evaluation of its behavior in response to a set of input against documented, required behavior.

12.3.3 Assessment

Assessment is determining the capability of a process through comparison with a standard. The exact methods used are dependent on the standard applied. Two standard assessment methods that are frequently employed are ISO 9000 and SEI SW-CMM. Malcolm Baldrige is another assessment standard, but is not used as often by software projects. Congress established the Software Engineering Institute (SEI) at Carnegie Mellon University in 1984 to improve the practice of software engineering. A key product developed by the SEI to aid in this mission is the Software Capability Maturity Model (SW-CMM). The SW-CMM is a model for software process improvement. The model establishes criteria describing the characteristics of a mature software organization and has staged software process maturity levels. There are 5 levels of process maturity, with level 1 being the lowest and level 5 being the highest. Within the maturity levels are groupings of software engineering topics called Key Process Areas (KPAs). The ISO 9001 international standard was established to address quality requirements across diverse industries. As such, the requirements within the standard are written in a generic manner to accommodate the diversity of applications. The corresponding ISO 9000-3 document gives guidance for applying the standard to software. Use of assessments may involve individuals outside of the organization such as a CMM lead assessor or an ISO registrar, but many times the assessment is conducted using internal resources to identify areas for improvement or in preparation for a formal assessment. Assessment uses a combination of random auditing and interviewing to answer a list of questions that is tailored to fit the organization being assessed.

12.3.4 Analysis

Analysis consists of performing the following:

- Causal Analysis and Defect Prevention Process
- Reliability Prediction
- Statistical Process Control

12.3.4.1 Causal Analysis and Defect Prevention Processes

The purpose of these activities is to address the process weaknesses that allowed product defects to be inserted in order to prevent reoccurrence of similar types of defects. One method to accomplish this objective includes root cause analysis and process brainstorming. First the team of individuals, which may include developers and other analysts, determines the root cause of the defect insertion. If the cause is systemic and/or may be repeated, brainstorming for a remedy is performed to decrease

the likelihood of reoccurrence of similar defects under similar circumstances. Ideas for process improvement are generated from the brainstorming session and passed on to a process management team. These activities may be performed at various stages of the software life cycle, but it is recommended that the elapsed time between defect discovery and this type of analysis be minimized.

12.3.4.2 Reliability Prediction

The IEEE Standard Glossary of Software Engineering Terminology definition of software reliability is: “The ability of the software to perform its required function under stated conditions for a stated period of time.” The ability to predict the reliability of a software system would enable project management to better perform product assurance and assess readiness for release. Three bases used in estimating reliability are failure record, behavior for a random sample of input points, or quantity of actual and “seeded” faults detected during testing. However, these methods are imperfect; software reliability prediction is still a science under development. Furthermore, this technique requires an extensive error history database.

12.3.4.3 Statistical Process Control

Statistical process control is the use of statistical methods to assure both process and product quality. These methods include Pareto analysis, Shewhart control charts, histograms, and scatter diagrams. This technique can be used to evaluate if a process is out of statistical control, thus indicating process defects and/or potential for increased product defects.

13. Software Test and Evaluation

Software Test and evaluation (T&E) is a significant element of the overall approach to verification and starts with identification of test requirements for each functional requirement that is to be satisfied with software, whether NDI (non-developmental includes: GFE, GFM, COTS, GOTS), or developed. Software system testing is required prior to integration of the software into the overall system. The software testing process should address any lack of compliance found by the T&E process. The resolution of a compliance issue discovered during test and evaluation could involve a change in allocated requirements, design, or life cycle plans.

A Software Test Integrated Product/Process Team (SW IPT) is to be established chaired by the cognizant SMC project element, with support of the Responsible Test Organizations and other Services stakeholders. The scope of this IPT is to develop and maintain the software testing portions of all *Test and Evaluation Master Plans* (TEMPs) and support the RFP development process by ensuring requirements are testable. This IPT is also responsible for addressing the development and application of software Modeling and Simulation in testing activities.

In addition, the SW IPT will provide continuous interface with the integrator community that will ensure that interface issues associated with applications not chosen for the demonstration program are addressed early in the spec and ICD development.

See Appendix C for a Testing checklist.

14. Metrics

Metrics are measurements of different aspects of an endeavor that help us determine whether or not we are progressing toward the goal of that endeavor. They are used extensively as management tools to provide some calculated, observable basis for making decisions. Some common metrics for projects include schedule deviation, remaining budget and expenditure rate, presence or absence of specific types of problems, and milestones achieved. Without some way to accurately track budget, time, and work progress, a project manager can only make decisions in the dark. Without a way to track errors and development progress, software development managers cannot make meaningful improvements in their processes. The more inadequate our metrics program, the closer we are to herding black cats in a dark room.

The right metrics, used in the right way, are absolutely essential for project success. Too many metrics are used simply because they have been used for years and people believe they might be useful. Each metric should have a purpose, providing support to a specific decision making process. Leadership too often dictates metrics. A team under the direction of leadership should develop them. Metrics should be used not only by leadership but also by all the various parts of an organization or development team. Obviously, all metrics that are useful to managers may not be useful to the accounting people or to developers. Metrics must be tailored to the users.

The use of metrics should be defined by a program describing what metrics are needed, by whom, and how they are to be measured and calculated. The level of success or failure of your project will depend in a large part on your use or misuse of metrics – on how you plan, implement, and evaluate an overall metrics program.

The SPO and developer should mutually agree on and implement selected software metrics to provide management visibility into the software development process. The metrics should clearly portray variances between actual and planned performance, should provide early detection or prediction of situations that require management attention, and should support the assessment of the impact of proposed changes on the program. See appendix C for the Measurement and Metrics checklist. The following software metrics are highly recommended, and the first three provide critical information to support lessons learned:

- Software Size
- Effort
- Schedule
- Requirements Definition and Stability
- Software Progress (Design, Coding, and Testing)
- Software Development Staffing
- Earned Value Management (Cost / Schedule Variance)
- Quality
- Development Tools and Laboratories Status
- Computer Resources Utilization / Reserve Capacity

- Software Maturity Matrix (Optional)

These indicators should be tailored and implemented consistent with internal developer systems. The applicable integrated product team (IPT) should implement additional indicators or measures to address software issues deemed critical or unique to the program. All software management indicator/measure information should be available to the SPO

14.1 Software Size

Software size is a basic measure that should be tracked over time by all programs involving software development or sustainment. Software size is the most influential factor on software development effort and schedule, which are key measures for all programs.

Software size can be tracked in the manner that best suits the program, whether in source lines of code (SLOC), equivalent lines of code, function points, or some other measure. For original software development, SLOC may be the measure most readily available. For sustainment where relatively small changes are being applied to large existing software products, or for development efforts that primarily involve the integration of existing software products, some type of equivalent lines of code measure may be more appropriate to identify and track the volume of effort required.

Typical categories of SLOC, which have varying effects on development effort and schedule, include the following:

- New SLOC: estimated or actual number of SLOC of newly developed software
- Modified SLOC: estimated or actual number of SLOC of existing software requiring change
- Reused SLOC: estimated or actual number of SLOC of existing software used as-is
- Total SLOC: the totals of the above three categories

Software size should be tracked down to the CSCI level (at least), and should also be broken out by spiral, increment, block, etc.

Whatever measure is used, there should be a clear definition of that measure so that it is understandable and can be consistently applied. Estimated software size should be captured and recorded at the time of contract award or start of development, and it should be continuously tracked throughout the development, since changes in size projections are good predictors of corresponding changes to the required software development effort and schedule.

At system / capability delivery, the actual software size should be captured and recorded.

14.2 Effort

Projected (estimated) software development effort is determined primarily from software size, but also depends on other factors such as developer team capability, tool capability, requirements stability, complexity, and required reliability. Effort is typically measured in staff hours or staff months and directly relates to program cost.

Estimated software development effort should be captured at the time of contract award or start of development, and it should be tracked over time, consistent with the method and level of detail used to track software size. The actual effort required should be captured and recorded at the time a capability (spiral, increment, block, etc.) is delivered.

14.3 Schedule

The projected (estimated) software development schedule is determined primarily from software size, but also depends on other factors such as developer team capability, tool capability, requirements stability, complexity, and required reliability. Schedule duration is typically measured in hours or months.

The planned software development schedule should be captured at contract award or start of development, it should be tracked over time consistent with the method used to track software size and effort, and it should be broken out at the same level of detail. Actual schedule durations should be captured and recorded at the time a capability (spiral, increment, block, etc.) is delivered.

14.4 Requirements Definition and Stability

The number of software requirements should be tracked over time, as well as changes (additions, deletions, modifications) to those requirements. This measure provides an indicator of how the requirements are maturing and stabilizing, and can be an early indicator of rework.

Requirements and requirements changes should be tracked down to the CSCI level, by spiral, increment, block, etc.

14.5 Software Progress (Design, Coding, and Testing)

Similar to the tracking of requirements, progress in design, coding, and testing should be tracked over time. Computer software unit (SU) can be tracked at this level, including original components or units designed, coded, or tested, as well as new additions, deletions, or modifications to the planned numbers in each of these areas.

14.6 Software Development Staffing

This measure is used to track the status of the developer's staffing level versus the planned levels over time. It reflects the total software personnel available as well as unplanned personnel losses and personnel experience levels. It can be broken down further into types of personnel, such as management, engineering, qualification/testing, and quality assurance.

14.7 Earned Value Management (Cost/Schedule Variance)

Earned value measurements should be applied to software development to provide objective measures of software effort and schedule status. Earned value management is perhaps the most effective method to predict effort and schedule problems in software development. Application of Earned Value Management System (EVMS) requires work package definition with defined events, objectively verifiable products, and established earned value and schedule for those products. Typical parameters that can be tracked include cost and schedule variance, cost and schedule efficiency, percent complete and percent spent, and estimate at completion.

14.8 Quality

Software quality can be tracked in several ways that include discrepancy reports (DRs) and defect density.

14.8.1 Discrepancy Reports (DRs)

Tracking of problem or DRs over time is one metric approach for reporting discrepancies. This may be done by categories, including total number of problem reports open, number open and unresolved, number open and resolved, number closed, etc. These could be further broken down by additional categories (for example, development phase in which the problem was discovered, development phase in which the problem was inserted) or by severity. This is another area where the measure should be tracked down to the CSCI level or lower, by spiral, increment, block, etc.

14.8.2 Defect Density

Defect density is another measure of quality. A typical way to measure defect density is to divide the number of problems or discrepancies by the software size in lines of code. This measure is typically expressed in defects per thousand lines of code, and is applicable to a particular software product, delivery, spiral, block, increment, etc.

14.9 Development Tools and Laboratories Status

This measure assesses whether the tools required by the developers, integrators, and testers are available when needed. One method of tracking this measure is to simply use a matrix to identify the tools of interest, the purpose, the environment, interdependencies, need date, available date, and general information on status. Tools that are candidates for such tracking include components of the System / Software Engineering Environment (S/SEE) or the system integration labs.

A second method of tracking tools and laboratories status is to track actual utilization over time for high-demand resources such as integration labs.

14.10 Computer Resources Utilization / Reserve Capacity

This measure is used to track, over time, resource memory and throughput usage and spare capacity for each computer resource, as well as I/O and multiplex bus capacity. Monitoring these parameters should aid in ensuring that the software design will fit within the planned computer resources, and that adequate reserve capacity is available to permit some level of enhancement in the post deployment support phase.

Computer Processor Utilization (CPU) is employed to track the percentage of CPU execution time consumed by the planned or actual software operating within the computer resource in a worst-case scenario. Memory utilization tracks the percentage of computer resource memory consumed by planned or actual software allocated to the computer resource in a worst-case scenario. I/O and multiplex bus utilization tracks the percentage of planned or actual I/O or multiplex bus resources consumed by system operations during worst-case operational scenarios.

15. Risk Management

Identify issues and risks associated with the software development, integration, and deployment. During the Requirements Analysis (RA) Phase of the Program develop an initial software risk assessment and, as far as possible, identify the means to reduce, eliminate, and mitigate the software risk factors throughout the program. Ensure that the RFP requires a Risk Management Plan (RMP) from the offerors. The program office and the contractor's program risk assessment and related management plan are living documents and must be continuously reassessed throughout the life cycle. The types of risks and their expected probability of occurrence and level of severity if they occur should be identified for the software, as characterized below. See appendix C for the Risk Management checklist.

15.1 Technical/Performance Risks

Some typical technical and performance risks for software could include:

- Timely and complete development of detailed software requirements
- Identification of software security requirements and their complexity. Timeliness, completeness, impacts to operational utility, and affordability of security requirements can adversely impact software development.
- The security requirements in the development environment can impact the software development activity
- Maintenance of independent software development and software test bed facilities
- Ensuring that the software development and test environment configurations stay consistent with the target operational environment and platforms
- Software system integration and its subsequent integration with hardware
- Planned technology advances and plans for technology insertion have risk.
- Requirements Instability as of unplanned requirements additions, deletions, and changes

15.2 Schedule Risks

Some typical schedule risks for software could include:

- Underestimating the complexity of the software and/or the integration, due to lack of information at a given point in time
- Integration often represents schedule risk – for software there can be two levels of integration – 1) the software into the "system" software and 2) the software into the hardware components/environment.
- Related to the integration risk, is the interface control risk. If software and system interfaces change or are not properly controlled, the software integration may be impacted.
- Requirements Instability as of unplanned requirements additions, deletions, and changes

15.3 Cost Risks

Some typical cost and risks for software could include:

- Unique or demanding performance and integration requirements
- Limited resources, which can include such things as the number and type of personnel, training levels of personnel, limited development or test facilities, limited tools and/or knowledge of tools
- Parallel development of hardware environment/equipment and software and/or software platform
- Changing funding allocations
- Requirements Instability as of unplanned requirements additions, deletions, and changes

16. Software Support

16.1 Software Support Planning Concepts

Software support activities occur during each phase of the software lifecycle. Therefore, post development software support planning must begin very early in the software development lifecycle. Ultimately, the ability to conduct successful post development software support depends on what occurs, or what fails to occur during the period of pre-deployment of developed software.

16.2 Post Development Support Planning Concepts

The overall support concept for the software implementation, deployment and maintenance should be described in a Software Lifecycle Support Plan. Support strategies and their affect on the software specifications and ICDs will be determined during the RA Phase and refined in subsequent program phases. The specific software support strategy will vary and be tailored to the specific program deliverable software.

16.3 Resource Identification Concepts

Resource analysis begins upon designation of the agency or organization identified to conduct post development software support. A detailed methodology for post development resource analysis and projection is contained in MIL-HDBK-347.

16.4 Software Support Planning

The project officer must make certain that adequate facilities, support software, personnel, and documentation are available after the development stage so that the software can be maintained in an operational condition. However, these facilities, personnel, and documentation will not be available unless a realistic Software Support Plan (also known as the Computer Resources Lifecycle Support Plan, CRLCMP) is prepared early in the software acquisition process and the plan is fulfilled.

The Program Office acquiring the system should prepare the Software Support Plan. The project officer should make certain that experts in the areas of software logistics support are assembled for plan preparation. This team is usually called the Computer Resources working Group (CRWG). The working group should be assembled very early in the acquisition effort, before contract award if possible.

One of the first major decisions of the CRWG is the selection of the organization that will be responsible for the software support after system delivery. The software can either be supported by the developing contractor or by a government logistics organization. The latter is called organic maintenance. Regardless of whether software support will be performed organically or by a contractor, the project officer should ensure that the Program Office acquires the knowledge, documentation, support software, and data rights that would enable the organic maintenance option to be exercised at a later date if required. Otherwise the government could be locked into an unfavorable dependence on the original development contractor.

If the chosen support concept is organic (government), the designated support organization needs the same support software as was required during software development. The project officer must ensure that the contract specifies the required software and hardware to support the software system.

16.4.1 Software Support Plan Content

The major topics covered in the Software Support Plan are described in the following subsections.

16.4.1.1 Organization

Who will be responsible for software support throughout the systems life? Who are the focal points in the acquisition and user organizations that work with the support organization?

16.4.1.2 Transfer of Responsibility

What conditions must be established before responsibility for software integrity passes from the acquisition to the support organization? How and when will transfer of responsibility take place?

16.4.1.3 Documentation

What documentation is needed to enable life cycle support? How and when will it be obtained? How will its integrity be evaluated?

16.4.1.4 Support Software Needs

What support software (e.g., CASE Tools, simulators, etc.) will be needed for life cycle support? How and when will it be obtained? How will its correctness be established and how will it be controlled?

16.4.1.5 Equipment Needs

What computers and other hardware (Software Engineering Environment--SEE) are needed? How and when will the SEE be acquired and maintained?

16.4.1.6 Personnel Needs

How many people are needed to staff the support facility? When are they needed?
What skills needed and how will these skills be obtained.

16.4.1.7 Configuration Management

What actions will be taken to establish the initial software's baseline configuration and to maintain the software's integrity throughout the system's lifecycle?

NOTE: A draft Support Plan should be prepared early, even if the details of the plan cannot be completed, so that the software development contract contains the essential provisions. The plan should be revised throughout the development effort, especially after key development milestones such as formal design reviews, as more information becomes available.

17. Commercial Off-The-Shelf (COTS)

A significant change to military acquisition that has been implemented during the last decade is the use of commercial components and assemblies that are already developed and available for sale in the public marketplace. The idea behind this change is, “If there is a commercial product already built that will satisfy our requirements, let’s buy it instead of developing our own.” This may require the lowering of standards so that the commercial product will meet the requirements. The following describes the key aspects of COTS:

- COTS is a product that is sold, leased, or licensed to the general public
- COTS is offered by a vendor trying to profit from it
- COTS is supported and evolved by the vendor, who retains the intellectual property rights
- COTS is available in multiple, identical copies
- COTS is used without modification of the internals

See appendix C for the COTS checklist.

17.1 COTS Benefits

There are several potential benefits that may be realized using COTS that include:

- Reduced or eliminated development costs
- Product Improvements paid for by vendor
- Reduced or eliminated development schedule
- Wide user base to prove the product
- Available skill base
- Industry investment in technology base
- Lower acquisition costs
- More component or vendor alternatives
- Development and maintenance time and cost savings
- Common “look and feel”
- System interoperability
- COTS-based development is software reuse
- The same component is used in many systems
- Many components are available; developers are responsible for selecting what they need

17.2 COTS Risks

While it may appear to be a case of “faster and cheaper” COTS components vs. “better” mil spec components, there are a number of risks that must be considered before deciding to acquire COTS instead of developing an item. The major concerns in using COTS include:

- Support (maintenance and logistics) may not be responsive enough to meet your requirements.
- Unforeseen environmental conditions may fall outside the COTS product specifications.
- There may be incompatibilities with hardware, software, processes, or the operational environment.
- Verification and validation effort and costs may be higher than anticipated.
- Integration may be more difficult than estimated.
- Training costs may be higher than for government-developed systems.
- Operation and maintenance costs may be higher than for equivalent government components.
- Vendor viability (technical proficiency, stability, dependency on other sources).
- Security questions may be unresolved. Is there hidden, malicious code, a “back door,” or easily bypassed security checks?
- Product volatility. Product changes are subject to the vendor’s choices and timing.
- Product quality may be lower than required, impacting reliability, safety, maintainability, and other considerations.
- Not all COTS components are of high quality -- robust, flexible, etc.
- COTS components may not meet our performance parameters.
- COTS products are frequently discontinued and replaced by a “new and improved” replacement and may not be compatible with your use.
- Products are not always well specified; it can be hard to understand what they really do.
- It can be difficult to integrate several COTS components.
- There may be licensing issues that affect how we can distribute the resulting system to our customers.
- Few guidelines are available for estimating the schedule and resource requirements for COTS-based development.
- Use of COTS makes the government dependent on the vendor.
- COTS vendor goes out of business or otherwise fail to support the product we’ve selected.
- Testing involving COTS is more difficult – There is usually no access to the source code, only the black box.

17.3 COTS Mitigation Techniques

In order to reduce the effects of risks involved with COTS products in an acquisition or development project, the following mitigation techniques should be employed:

- Thoroughly understand the requirements of the system to be built
- Use good systems engineering practices, i.e., understand the functions the COTS software is to perform and the necessary interfaces with the remainder of the system
- Gain knowledge about the marketplace and vendors. Know which are viable
- Learn about the products, i.e., the functions performed and the required interfaces, to be able to make informed judgments
- Understand which things must change the least and which things are likely to change the most
- Know all the options
- Conduct a make vs. buy vs. rent trade study
- Reduce integration and other issues by designing around a major COTS product
- Employ industry standards wherever feasible
- Establish a robust verification plan and environment
- Involve the vendor throughout the life cycle
- Get product or vendor certification if possible
- Have vendor put source code into “escrow” for future needs
- Consider a product line approach
- Establishment of COTS Vendor Relationship (It may be best to select a few key vendors and establish strategic partnerships) to address:
  - Favorable pricing
  - Improved support
  - Enhancements
  - Support for older versions
  - Benefits vs. requirements flexibility
  - Risks and how they’re handled
  - Licensing concerns
- Scope of Licensing Issues to consider are:
  - Transferability of the license
  - Should not contain software “time bombs” (. e.g., If you don’t do something, the software will shut down or the vendor has a “back door” into their software, etc.)
  - Look for enterprise-wide license opportunities
- Investigate licensing alternatives
- Use licenses to incorporate special provisions, such as:
  - Vendor commitment to inclusion of modifications in the next commercial product release
  - Support services for integration
  - Assurances regarding product splits

18. Reuse Software

Software reuse is the process of implementing or updating software systems using existing software assets. Software assets, or components, include all software products, from requirements and proposals, to specifications and designs, to user manuals and test suites. Anything that is produced from a software development effort can potentially be reused.

A good software reuse process facilitates the increase of productivity, quality, and reliability, and the decrease of costs and implementation time. An initial investment is required to start a software reuse process, but that investment pays for itself in a few reuses. The scope of software reuse includes the development of a reuse process and repository (produces a base of knowledge that improves in quality after every reuse), minimizing the amount of development work required for future projects that can use that component, and reducing the risk of new projects that are based on repository knowledge.

18.1 Prerequisite to Creating Reusable Software

The following covers the criteria that needs to be addressed to implement a software reuse approach:

- A Reuse Strategy
- Reusable Components
- Allow for a longer Design Phase to allow for refining the reusable components and thus reducing rework
- Test/Validation Suite
- Technical Data Package
- Development/Sustainment Environment
- Plan for a thorough Regression Test at predetermined points in the effort
- Make sure that Field Upgrades are addressed early in the effort
- A Configuration Management Tool is Essential
- An incentive fee for weapon system reuse
- Relationship Management – The Vendor, the customer, and the developer must be in the partnership together.
- Management Support – Management must support the additional resources required for reuse.

18.2 Reuse Software Benefits

There are several potential benefits that may be realized to reuse software that include:

- Ultimately will achieve shorter development times
- Reduced documentation size (i.e. A 1999 Missile Program reduced documentation by 37.7% due to reuse).
- For a Common Missile Program in 1999: 77% of the code was reusable, 34.8% of the design was reusable, and 89.4% of the documentation was reusable.
- Reduce the rate of ever increasing Software Costs
- Have higher project success rates
- Improve application performance
- Increased productivity
- Reduce schedule
- Enhanced quality

18.3 Software Reuse Risks

The major concerns in using reuse software include:

- Cost -This issue can be addressed by Incentives and Contract Terms
- Legal Issues such as Warranties, Liabilities, and Royalties -These legal issues can be addressed through Contract Structuring
- Technical Barriers such as lack of consistent methodology
  - These barriers can be reduced through the use of Architectures and a Metrics Framework
  - A Metrics Framework can be used to identify software code components that have a potential for reuse.
- Competitive Edge (e.g., Need to keep things hidden)
  - Difficult to work around
  - Organizations are unwilling to Share Information (Design, Source Code, Test Code, etc).
  - This would give away their competitive edge

Need to determine who is responsible for the code

- Additional cost in people and training
- It is difficult to estimate the cost of reuse up front
- There is a training and support mindset

18.4 Liabilities of Reuse

Some of the shortcomings of reuse software include:

- The initial cost of development with reuse in mind is higher
- Extra costs to rework reusable components throughout program
- There are going to be Common Architecture Tradeoffs (some good and some bad)
- When you find a “bug” in the reused code, you may have common bugs across multiple systems

Note: a common mistake is to go think that software reuse is “The Silver Bullet.” That is will allow you to produce software better, faster, and cheaper (The 3 pillars of software development). You might be able to produce the software better with higher quality and faster, but probably not cheaper.

19. Independent Verification and Validation (IV&V)

The purpose of this section is to provide the Project Officer insight into how the need for a Software Independent Verification and Validation (SIV&V) contractor is established, the scope of the SIV&V effort relative to the size of the project itself, what the content of the RFP (Section L and M) should be, what CDRL items to call for, what to look for in source selections, and how to manage the SIV&V contract.

19.1 What SIV&V Accomplishes

The Software Acquisition Project Officer must have a clear picture of what SIV&V is to accomplish for the program and when it is appropriate to use SIV&V. SIV&V spans activities throughout the lifecycle of a software systems acquisition. The SIV&V contractor can review specifications prepared by the developer to check the allocation of requirements; perform requirements traceability between all levels of specifications; participate in design reviews and independently test the software implemented by the developer. The purpose of using an SIV&V contractor is not to avoid the occurrence of all software problems, but is to ensure the performance, integrity, reliability, safety, supportability, and quality of the final system software. These critical software requirements will be optimally impacted when software IV&V activities are integrated with the software development process. The SIV&V activities must be performed in parallel with and complement the software development process to enhance the early exposure of requirement inconsistencies, design and coding errors, and thereby improve system software quality.

19.2 Establishing the Need for SIV&V

Before establishing a SIV&V program it is first necessary to determine whether the software system being considered warrants an IV&V effort. The consequences of software errors establish whether or not SIV&V is indicated for your program. If there is some chance that an undetected error could cause loss of life or personnel injury, or jeopardize mission success, SIV&V is required.

Determining estimates of how much of the project should undergo SIV&V and the cost depends on where you are in the lifecycle. SIV&V can span the lifecycle, performing requirements analysis, design, code, integration, and test.

19.3 Establishing the Scope of SIV&V

The SIV&V effort needs to be tailored to and commensurate with the criticality level of the software being developed. After determining the need for SIV&V, its scope can be established by performing a criticality analysis on the software. Preferably this should be done at the Software Configuration Item level using the Software Requirements Specification and Interface Requirements Specification. However, this level of detail is not always available. In this case the analysis can be performed at the System Specification level.

Regardless of whether you use the Systems Specification or the Software Requirements Specification the criticality analysis of the software requirements performed using the following six-step procedure:

- List the software requirements.
- Identify the potential impact of undetected software errors or faults on the software requirements.
- Estimate the probability of occurrence for each error.
- Calculate the requirements criticality value.
- Calculate the overall criticality value for the system or for each Software Configuration Item.
- Select the appropriate level of SIV&V based on the systems Software Configuration Items criticality value.

Note: Since we feel that it would be outside the scope of this document to get into the details of "How To" do the criticality analysis of the software requirements we advise you to work with your FFRDC and/or SETA contractors on doing the actual analysis.

Once the criticality values for the target SCIs have been computed the criticality analysis has been completed and the appropriate level of SIV&V can be assigned. The criticality values are then used to select one of three levels of SIV&V effort. Each SIV&V level is progressively more detailed and comprehensive than the previous level.

19.4 SIV&V Tasks

In this section we will identify SIV&V tasks for the three levels and relate these to the software development phases. The three SIV&V levels include various tasks that increase in scope and complexity from one level to the next.

SIV&V involves activities over the entire software development lifecycle; it's more than just another software test activity. Too often, the SIV&V is not started until the software is in the test phase, when problems become most visible. If you wait for testing to identify and remove problems, you lose most of the cost leverage associated with SIV&V.

Integrating the SIV&V activities with the software development lifecycle helps ensure the earliest possible detection of requirement inconsistencies and software coding errors. The tasks of the SIV&V contractor are synchronized with the tasks of the software developer so that the contractor can analyze the products and processes as soon as they become available. In addition, tying the two activities together helps the project officer tailor the SIV&V effort to the software development effort. The two efforts, although independent, can compliment and reinforce, rather than duplicate or hinder each other.

19.5 Termination of SIV&V Tasks

The tasks performed by the SIV&V contractor can vary with costs versus benefit. The project officer should terminate the SIV&V task when the cost exceeds the benefit. Criteria or thresholds must be established for terminating SIV&V support. If the SIV&V effort is for level 3 tasks, the termination criteria may be completion of reviews on the operations and maintenance manuals. Include termination procedures for all SIV&V efforts.

19.6 Estimating SIV&V Costs

There are no standard formulas for estimating SIV&V costs. SIV&V cost estimating requires a thorough understanding of the systems software and sound engineering judgment. The following section provides guidelines to follow when estimating SIV&V cost, based on historical trends and past experience.

19.6.1 Range of Estimates

The cost of software IV&V can range from 10 to 50 percent of the cost of developing the system software, depending on the IV&V level selected. However the majority of the IV&V programs are usually at the lower end of this range.

19.6.2 Cost Factors

Assuming that the software development cost has been correctly estimated and accurately defined. Variations in cost factors used for the estimation can impact software development cost and later, IV&V cost. At a minimum the following will be considered:

- Size (Lines of code and function points)
- Complexity
- Volatility of requirements
- Use of standards
- Programming language
- Quality of existing software and documentation (when modifying existing software)
- Maturity of systems software and development tools
- Programmer experience level
- Training considerations

19.7 Selecting an SIV&V Agent

The Project Officer must make sure that the IV&V contractor is independent from the software developer. For the IV&V contractor to maintain its objectivity they should be independent from the prime contractor or software developer.

19.7.1 Selection Factors

The following guidelines should help the Project Officer select the best-qualified IV&V contractor:

19.7.1.1 SIV&V Experience

SIV&V involves the use of specialized techniques performed by highly skilled personnel to detect errors not usually found by the software developer. Therefore, one of the most important qualifications of a potential IV&V agent is experience. The IV&V agent must have a strong background in the latest IV & V techniques and have a successful track record. A large amount of IV & V experience gives the IV&V agent the intuitive in- sight into where problems are most likely to occur, the ability to quickly recognize problem areas and pitfalls, expeditiously recommend sound solutions, and experience in working effectively with the software developer in a pressure-filled environment. The agent's IV&V experience must also be commensurate with the level of IV&V desired on that application.

19.7.1.2 Application Experience

The IV & V agent must have experience in developing or performing IV & V on similar type systems and be qualified in the technology area under development.

19.7.1.3 Personnel Experience

The management and technical personnel must have a strong background in conducting IV & V on programs similar to the one in question and be proficient in using their company's IV&V tools. In addition, they must also have a detailed knowledge of similar systems and the technology areas being developed.

19.7.2 IV&V Tool Library

The IV & V agent should have IV & V tools that are appropriate for the specified IV & V level. These tools should have been regularly used in the past and not require inordinately specialized talent to use. The existence of a large tool library is of little value if the tools cannot be easily used or the IV & V agent is not thoroughly familiar with their use and limitations.

19.7.3 Contractor Qualifications

To determine the qualifications of an IV&V contractor the Project Officer has two options. The first is to review the contractor's past performance on similar applications. The second is to perform a capability evaluation during the pre-award survey. By tailoring the pre-award survey, the Project Officer can get a quick look at an IV&V agent's potential.

20. Training and Experience

This involves making sure that personnel have appropriate training and experience for their role on this acquisition. Program Office should determine the minimum Acquisition Professional Development Level or APDP. Areas and levels to include: Program Management Level I-III, Systems Planning, Research, Development and Engineering – Systems Engineering Level I-III, Contracting Level I-III, Acquisition Logistics I-III, Test and Evaluation Level I-III, Financial Management Level I-III, and Product Quality and Manufacturing level I-III.

See Appendix A Lessons Learned on pitfalls.

21. Software Acquisition Lessons Learned/Best Practices

The SW IPT will leverage off of the software lessons learned experienced by the software development, production, and fielding of earlier SMC programs. One of those lessons learned was the importance of early involvement of industry and National Security Agency (NSA) in the RA Phase of the program to reduce technical and integration risks. The Project Office will hold Industry Open Forums and solicit feedback from industry throughout the RA Phase to ensure early industry involvement and feedback in the approaches, risks, and costs to meet the Users' requirements.

In addition, there should be User Forums to incorporate the Services' CDDs or host platform unique requirements that may have to be reflected in the specifications and ICDs and ensure they are mutually understood.

See Appendix A Lessons Learned for complete overview of software acquisition pitfalls.

Appendix A Lessons Learned

This appendix discusses software acquisition management lessons learned by the authors of this Handbook and other experts who have multi-decades of experience in acquiring and developing large-scale DoD software systems.

These lessons learned are primarily for SMC project managers executing their duties as Project Officers, contracting officers, systems engineers, and other managers involved in software acquisition. These lessons learned are not inclusive but presents a short list of software acquisition universal truths based on lessons learned, and often re-learned, on numerous large-scale development efforts over the course of many years (decades).

The discussion of each universal truth is broken down into four sections identifying, explaining, and emphasizing what every successful software project manager should understand. The information is provide in the following format:

- The Universal Truth
- Why this Universal Truth is important
- Benefits of this Truth
- Specific ways that you can emphasize this universal truth within your own program

A.1 Users - Understanding and Managing Their Expectations

The Universal Truth: Involve users early, regularly, and often in a managed way.

A.1.1 Why This Is important

Involving users of the system in the acquisition from the beginning:

- Provides a greater likelihood of mission success
- Their involvement is critical to understanding both the requirements and the operational environment in which the system must work.

Note: But be careful that users involvement does not lead to runaway costs; “it is all too easy for users to ask for the moon when they do not have to pay for it”.

In order to most effectively implement a system to the user needs:

- Take time to understand how your users create and add value to their mission.
- Ensure that your project enhances their ability to perform and those they buy-in to what the project will enable them to do.
- Beware of users focused on “what was” - keep them focused on the “what must be”.
- The longer the development cycle, the more likely it is that the users will reject the system when delivered.

Remember, you are trying to hit a moving target; the user base changes and user expectations are constantly remolded by their experiences with commercial products. The situation is exacerbated when you are developing a replacement system because your antecedent will continue to evolve long after you must freeze your baseline.

A.1.2 Benefits

The benefits derived from involving users early, regularly, and often in a managed way are:

- Greater likelihood that the program will match the needs of the user community.
- Greater likelihood of stabilizing requirements early in the development cycle.
- Greater likelihood that performance trades will have the least impact on the user mission.
- Greater acceptance when performance trades are required due to cost and schedule constraints.

A.1.3 How Best To Emphasize

The ways to best emphasize to the user:

- Use joint application information management sessions early in the process to obtain input. These sessions are Integrated Product Teams designed to translate user needs into requirements in a manner both users and developers will recognize.
- Develop a concept of operations (CONOPS), written in user language, before beginning the development (including scenarios and use case diagrams, if appropriate).
- Update the CONOPS as your user environment changes. Demonstrate your commitment to the user by controlling the CONOPS as you control requirements.
- Map the CONOPS to your top-level requirement specifications.
- Develop and stabilize the user interface early (e.g., prototype) and evolve the functionality to support it.
- Select user representatives carefully and avoid having too many cooks (at a minimum, user reps must have recent experience and be respected in their community).
- Have a process by which the users prioritize their requirements.

A.2 Your Development Team - Selecting and Getting the Most Out of Them

The Universal Truth: Watch what they DO (contractors) not what they say. This also applies to management both the contractor's and yours.

A.2.1 Why This Is important

Your goal is to achieve tangible, measurable results, not just well sounding promises. Past performance on related or similar efforts is still one of the best ways of improving your chances of a successful development.

Assess the contractor for the following:

- Make sure the winning development contractor has extensive experience with the business domain as well as the proposed development methodology, hardware, software languages, and tools prior to start of contract.
- Do not let your contract become a training environment for the contractor's team.
- Focus on understanding your developer's culture (cowboy attitude can kill you).
- Look for evidence of mature engineering processes.

- Ensure that the right expertise for the program is available and committed.
- Take time to understand your developer's methodology and environment.
- Make sure the contractor treats their employees well (don't underestimate how hard it is to hire and retain good people).
- Remember, when you contract for software development, you are contracting for both people and process.

A.2.2 Benefits

The benefits derived from getting the best from your development team are:

- Resources dedicated to development and not to indirect activities such as training.
- A team that is focused on the same objectives and understanding the same constraints.
- Greater assurance that the team already has the skills it needs to be successful.

A.2.3 How Best To Emphasize

The ways to best emphasize to the contractor:

- Write the RFP carefully and emphasize past performance and relevant technical experience in the award criteria.
- Specify a dedicated development team in your Statement of Work.
- ·Require an oral presentation during source selection by key members of the proposed technical staff.
- Check every past performance reference you can.
- Plan and budget for the entire effort (e.g., Systems Engineering, Test, IV&V), not just the development phase.
- Be careful when you include Operating and Maintenance (O&M) on the same contract and avoid level of effort work.
- Insist on an accurate schedule and stick to it. If the schedule begins to slip, review the accuracy of the original plan and take corrective action.
- Insist on a detailed Work Breakdown Structure (WBS). Make sure it is well maintained and correlates to your reported earned value metrics.
- Don't get caught up in being a zealot for a particular computer, operating system, software language, or development process. Competent technical staffs with relevant experience are key to success.

A.3 Requirements - Establishing and Controlling

The Universal Truth: If it is not written down you will never receive it (verbal promises do not last past the first speed bump).

A.3.1 Why This Is important

Not being able to establish and control requirements is the underlying key destroyer of a program's success as follows:

- The lacks of a firm requirements baseline at the inception of the development effort or a progressive creep in the requirements baseline are major contributors to the failure of many software developments.
- Requirements often change, but they must be controlled and managed against the impacts to cost and schedule.
- Late changes to any type of requirement will have costly impacts, especially the further into the development cycle that these changes are made. Requirements control is essential from the acquisition-planning phase through completion.
- Emphasize both requirements definition and interface definition.
- Always ask the question: Where does it say, "I have to do that?" before accepting new (or even modified) requirements.
- Custom code can be just as inflexible as hardware. Do not believe that because it is software, you can accept requirement changes later in the design/development cycle than you can with hardware systems.

A.3.2 Benefits

The benefits derived from the establishment and controls of requirements are:

- Establishes a clear and concise definition of what must be accomplished (contractual requirements) that is understood by the entire development team.
- Ensures requirements are unambiguous, traceable, verifiable, and documented.
- Defines the scope of the development effort.

A.3.3 How Best To Emphasize

The ways to best emphasize to the establishment and control of requirements:

- If your organization does not have a written policy or procedure for defining and managing requirements, write one. If it does, follow it.
- Designate a group to formally oversee requirements definition and management. A good candidate for this is the test team. Ideally, this group can help ensure requirements are written in a quantifiable way that can be verified.
- Make sure your requirements management team is trained and uses established plans and procedures.
- Specify performance requirements with future growth in mind.
- Maintain a mapping of requirements to intermediate products and individual configuration items.
- When you need to change (add, delete, modify) requirements, document the change and track volatility (% change by month).
- Be willing to prioritize your requirements and drop from the bottom of the list when you are forced to make performance trades.
- Establish and never lose sight of the commitment to deliver a meaningful and useful product.

A.4 Risk - Identifying and Controlling

The Universal Truth: The risks you manage determine the quality you deliver.

A.4.1 Why This Is important

Software development tends to be one-of-a-kind. This means that, by its very nature, software development is a high-risk endeavor. You will need every imaginable control in place to manage your software development project, and this means actively seeking out risk items for consideration. If you are not thinking about what could go wrong, something surely will. You need to establish and aggressively a risk management approach to actively mitigate significant risk elements. You should identify any significant new items/technology/approach as a risk until proven otherwise. Remember, cost and schedules are always risk items until the project is finished.

A.4.2 Benefits

The benefits derived from proactively managing risk are:

- An active risk management process keeps risk identification and mitigation in the forefront of your thinking.
- A common rating scheme for risk quantification gives a prioritized list of what risks to tackle first (risk = consequence x probability). You will not know every program risk yourself.
- Getting everyone involved in the risk identification process will uncover risks before they become problems that are too difficult or expensive to solve.

A.4.3 How Best To Emphasize

The ways to best emphasize risk management:

- Use a formal risk management process.
- Ensure your risk management plan includes both a risk assessment and risk control portion.
- Update both systematically and frequently and be flexible.
- Re-evaluate each risk regularly.
- Attend to each risk mitigation strategy until the risk is reduced or eliminated.
- Integrate risk into the development process.
- Get full buy-in on the process from the team and do not let risk management become pro-forma or an excuse to call a regular meeting.
- For risk identification, use a process that is open to the entire development team.
- Decriminalize the risk identification process. Create incentives for developers to identify risks early through open team communications; develop two-way trust.
- Do not hammer them on their award fee when they keep you apprised of unknowns. Reward them!

A.5 Cost and Schedule Management

The Universal Truth: If you do not where you are going any plan will get you there.

A.5.1 Why This Is important

It all starts with a reliable and technically justifiable plan. Remember, if your developer does not have a plan that realistically reflects how the system can be built, any measurements made against that plan cannot be trusted to provide meaningful information. The key cost/schedule considerations that can cause major problems are:

- Unrealistic and overly optimistic earned value reporting can mask problems and delay corrective actions.
- Non-development procurements often distort earned value because of the way they are billed and paid.
- Use other metrics such as staffing profiles, use of overtime, etc., to validate what earned value appears to be telling you.
- Earned value is only one of several tools available to you it is not “the tool”.

A.5.2 Benefits

The benefits derived from proactively managing cost and schedule are:

- A good plan and accurate reporting go a long way toward making your job easier. With these two elements in place, your decisions are based on an accurate representation of what work really is being accomplished against a realistically achievable plan.
- You will achieve the most benefit from a good earned value system when you use it to look for and discover defects early.

A.5.3 How Best To Emphasize

The ways to best emphasize cost and schedule planning:

- Develop a metrics-based cost and schedule baseline that integrates technical accomplishment into a baseline plan and has as many discretely measured milestones as possible.
- Conduct an integrated baseline review shortly after contract award.
- Never write a Level-of-Effort (LOE) contract for a task that requires something to be delivered.
- Progress within software development tasks is notoriously hard to measure. Software elements, with few exceptions hang at the 95% complete stage, so use an earned value system with as close to a 0-100% earned value basis as possible. (0-100% is where earned value is given only when the task is complete.) Never allow a 100-0% earned value method!
- Identify and correct problems as you go. Establish a peer review process that goes hand-in-hand with the earned value system. The process should be in effect from the requirements analysis phase to the integration and test phase.
- Think of discovered variances as your program management friend. Decriminalize variance reporting from the beginning for a profoundly positive impact on the overall success of your project.

A.6 Configuration Management – Using To Control the Program Baseline

The Universal Truth: Shortcuts in software configuration management will cause infinite chaos “when the rubber meets the road”.

A.6.1 Why This Is important

Configuration Management (CM) is the cornerstone of a solid software development process, especially as the size of the software development grows. Large software systems are so technically complex that without complete and absolute control over the environment, the in-process product, and the requirements baseline, it is virtually impossible to prevent/detect problems until the later stages of development (integration and test).

CM is more than the golden copy once development is complete. It is a completely integrated process of identifying and defining the systems product set at established milestones and controlling the release and change of these items throughout the system lifecycle. It also includes recording and reporting the status of product items and change requests and documenting the completeness and correctness of the product items. To have a truly effective
CM program, it must be an in-line process.

A.6.2 Benefits

The benefits derived from proactively address configuration management:

- Allows retrieval of previous working versions and provides an audit trail of modifications.
- Provides tracking of defects by individual configuration items, thereby:
    - Improves overall product quality by identifying problems early.
    - Allows systematic tracking of changes, thereby reducing rework.
    - Prevents uncontrolled, uncoordinated changes and aids immeasurably in problem detection and correction.
- Use review boards to define and enforce CM procedures. Be able to trace defects from the first report to final disposition.
- Allow NO exceptions to established CM procedures. Make sure everyone uses them, including developers, test teams, system engineering, and YOU.
- You have a good CM program if you know exactly what application version, which HW/OS, which database, which defects, and which patches are at each workstation at any given time. If you do not, re-read this universal truth again.
- Encourage frequent working builds during development.

A.6.3 How Best To Emphasize

Anything that is shared by two or more people should come under configuration control.

A.7 Metrics - Making Decisions Based On Performance Measures

The Universal Truth: Base your decisions on performance measures (metrics). If it is not measured, you probably do not care.

A.7.1 Why This Is important

Performance measures should provide the project manager quantitative insight into the program's health. With this quantifiable insight, you will have a solid foundation to make management decisions. Performance measures should be based on the risk identified and tailored to the mitigation strategy selected. There are literally hundreds of performance measures associated with software development. Many of these are focused on the needs of your development team.

As the Government manager, the decisions you face are similar, but not identical. The focus of the Government manager needs to remain on the impact of cost, schedule, and technical performance. Performance measures allow you greater insight into the root cause of cost, schedule, and technical problems. This insight does not necessarily help quantify the associated impact or specify the action you should take. Know your management's risk tolerance and ensure your performance measures help keep you within that threshold.

A.7.2 Benefits

The benefits derived from measured performance:

- Objective status on cost, schedule, and technical performance.
- Early problem indication and reporting thresholds.
- Early indications of design or development problems.
- Objective thresholds for making decisions.

A.7.3 How Best To Emphasize

Key ways to emphasize performance measures:

- Develop performance measures using the goal question- metric method. This method bases performance measures on program objectives and phrases those objectives as quantifiable questions.
- Specify the tracking and analysis of performance measures in the RFP.
- Establish clear performance thresholds for each performance measure.
- Performance measures must be directly related to the risks you identify and support the actions you take to mitigate those risks.
- Know exactly what decisions you need to make when performance thresholds are exceeded.
- Establish (if your management has not already directed it) reporting thresholds for each measure or combination of measures you select.
- Establish acceptable "watch and danger" thresholds for each measure as a function of time. For example, requirements volatility should reach zero by the Preliminary Design Review (PDR).
- If you do not know what decisions a measure supports, then THROW IT OUT!

A.8 COTS - Making Modifications to Commercial Software – NEVER!

The Universal Truth: Never allow software developers to modify commercial operating systems or other COTS products.

A.8.1 Why This Is important

Making modifications to commercial software causes endless delays, cost overruns, and general mayhem as software packages evolve. Once modified, each evolutionary release causes unforeseen breakage in the software baseline, generating a blizzard of problem reports and setting back development schedules. The pain continues long after the project transitions to operations.

A.8.2 Benefits

Leave will enough alone!

A.8.3 How Best To Emphasize

Key ways to emphasize:

- JUST SAY NO!
- This is an absolute prohibition. Violators should be prosecuted to the fullest extent of the law!
- Never pay a COTS vendor to specifically tailor or modify their product for you.
- Corollary: Never allow your developers to obtain source code for any COTS product.

A.9 COTS – Incorporating Commercial Products Into Your Development

The Universal Truth: COTS products are not necessarily the best solution. They bring as many risks as benefits. Understand both and make sure your developer does to before adapting.

A.9.1 Why This Is important

COTS manufacturers are driven by first to market concerns, the COTS product you thought you would get is seldom what actually is delivered. Building your own development around it is always a risky prospect. However, a judicious use of COTS can make a difference in your projects success.

Make your choices wisely, and keep in mind the following:

- COTS may cost you as much over the lifecycle as custom code.
- Integration of COTS is a non-trivial task. The difficulty goes up exponentially with the number of COTS products.
- Even the best products cannot be used out of the box except, perhaps, in a standalone mode.
- You will not be able to substantially influence or depend upon the evolution of COTS products.
- COTS products evolve asynchronously with one another and with your needs. You will not be able to freeze your COTS baseline for very long. Allow ample time in your schedule and budget to accommodate this.

- Expect to live with bugs in COTS products indefinitely. You will not be disappointed.

A.9.2 Benefits

- May be a cheaper and perfectly acceptable technical solution.
- Frees your developers for custom code.

A.9.3 How Best To Emphasize

Key ways to emphasize:

- Consider parallel developments and prototypes for COTS products that are not yet available.
- Investigate thoroughly the pricing structure of the product before buying. Site licenses or concurrent user limits are usually more cost-effective than seat licenses. Some products are priced attractively but cost a fortune to maintain.
- Select established products with large installed bases to increase the likelihood they will still be there when you deliver.
- Make sure your budget covers the complete cost of integrating and maintaining COTS products.
- Fly before you buy. Insist on a trial period for a COTS product and TEST, TEST, TEST to see what it does and does not do. Include labor cost and schedule to cover the trial period.
- Adapt your requirements to COTS capabilities. Wrapping the COTS product in custom code to improve or expand its functionality is EXTREMELY RISKY.

A.10 Unproven Technologies – Avoiding in Development

The Universal Truth: Cutting edge products lead to bleeding edge products.

A.10.1 Why This Is important

Cutting-edge technology, by definition, is out in front. That means there is a high probability of:

- Schedule delay
- Lack of trained resources to use the new technology/methodology
- Few CASE tools that support the product

Interfaces are undefined and may remain open long past the time when they should reasonably (and inexpensively) be closed. Further, this is a most uncertain time for the technology vendor. If the product is not an immediate commercial success, its longevity and promise for the future are at risk.

Few large-scale development projects can react quickly enough to survive the pace of change associated with cutting edge technologies.

Do not be a technology junkie! Think conservatively before you leap!

A.10.2 Benefits

Avoiding cutting-edge technologies will have significant cost savings over the lifecycle of the system.

A.10.3 How Best To Emphasize

Key ways to emphasize:

- Use only established technology, methodology, programming languages, and products.
- If necessary, reduce requirements/performance to avoid cutting-edge technology. If that is not possible, select products that espouse adherence to industry standards and open architectures.
- Make sure you have or add margin (schedule, cost, and performance) to the program (it will cost more, take longer, and still do less than originally intended).
- If you believe you must use them, carry unproven cutting-edge technologies as your No. 1 risk item.
- Track their development continually and intimately and you will avoid unpleasant surprises.

A.11 Upgrades - Hardware and Operating System

The Universal Truth: Hardware and software baselines (as well as operating systems) will become obsolete and require upgrading during the development cycle. GET OVER IT!

A.11.1 Why This Is important

Since hardware and operating systems performance is always increasing:

- Expect to double the processing capacity of the proposed platform by the time you deliver in order to achieve the required performance.
- For every 18 months of development time throughout the life of the program, expect to migrate to a newer version of hardware and operating system.
- Allow time in the schedule and cost in your budget.
    - Because of this periodic migration, defer procurement of the deployed system as long as possible to avoid delivering last year's great idea.
- Ensure your development approach can accommodate last-minute upgrades.

A.11.2 Benefits

With sound planning, you might actually deliver a system that is not obsolete.

A.11.3 How Best To Emphasize

Key ways to emphasize:

- Plan for upgrades in your cost, schedule, and development approach. It. is a fact of life.
- Pick a reliable COTS vendor.

A.12 Reuse Software – Incorporating In Your Development

The Universal Truth: Re-use costs extra in the initial development in terms of dollars and schedule as well as to the user in terms of system flexibility.

A.12.1 Why This Is important

Software re-use can reduce costs when re-use is included in the initial planning. Although typically associated with object-oriented technologies, the concept has been successfully demonstrated with other design approaches. Much has been written about the methodology for re-use in an object-oriented design. For other design methods, the probability of successful re-use is inversely proportional to the size and complexity of the re-used module. It is most successful for algorithms based on unchanging physics such as ephemeris work. The intent for re-use must be designed into the original version of the module.

A.12.2 Benefits

- Potential savings in development time.
- Potential savings in cost.

A.12.3 How Best To Emphasize

Key ways to emphasize:

- Intent to re-use code should be in the initial plan, not a desperate grab at a lifeline.
- Explicitly document the benefit, including the magnitudes and limitations, prior to committing to re-use.
- Manage re-use as a risk item, formally!
- Check other examples of re-use on similar projects.
- Make sure maintenance support is available and responsive enough to support your development needs. Avoid self-maintenance by your development team; this will cause divergence from the golden copy and you will have custom (not re-used) code.
- Investigate the stability of the current versions.
- Do not commit to create reusable code unless your project has the schedule and budget margin and the commitment of your management.
- Be careful with the amount of special code or glueware that is required to incorporate re-used code.
- Reward the developer for effective re-use.

A.13 Automatic Code Generators – Using to Cut Development Time

The Universal Truth: Avoid use automatic code generators where timeline performance is critical.

A.13.1 Why This Is important

Code generators are beloved by developers, especially those who tend to focus on lines of code as a productivity measure. Code generators offer the apparent prospect of easy modification; but by their very nature, they produce excessive (read inefficient) code, creating significant performance problems that you may not be able to overcome with

more powerful hardware. They also contain problems of inflexibility and constraints. Elect to use them only after you understand the risks and have determined that the risks are acceptable.

A.13.2 Benefits

The benefits may be the potential savings in development time.

A.13.3 How Best To Emphasize

If, despite your best effort, the developing contractor insists on using code generators for components that are critical to timeline performance requirements:

- Insist these algorithms be prototyped using the code generator, and conduct performance benchmarking tests during the design phase.
- Test for compliance with timeline performance requirements as early as possible.
- Allocate cost and schedule for reworking code to bring the performance into compliance with requirements.
- Require detailed allocation of timeline performance requirements down to the unit level.
- Account for tuning of automatically generated code to enhance performance.

Note: The same applies to code generated by humans (or programmers), but the need is even more critical for code generators.

A.14 GFE – The Pitfalls of Using in Your Program

The Universal Truth: Government Furnished equipment shifts the risk from the developer to the government. Leave the risk with the developer.

A.14.1 Why This Is important

The down side using GFE:

- The Government is generally not equipped to conduct large-scale procurements in support of development.
- Missed deadlines, short suspenses, and logistical nightmares tend to be the norm.
- Whatever the potential savings may seem to be, the real savings are generally nonexistent.
- Unfortunately, most GFE delays that do arise hit you late in the game when delivery schedules are critical and program delays are imminent. And it is YOU who are left holding the proverbial BAG.
- Many a development effort has been delayed by late or incorrect GFE.

A.14.2 Benefits

The benefits may be the potential savings in development cost.

The internal software allocated baseline is established after PDR with the SCCB functioning as the change control authority over the software requirements specifications. The functional baseline is established following the completion of the System Design Review. The authenticated functional baseline consists of System Specification CDRL.

A.14.3 How Best To Emphasize

- Negotiate reduced fees on costs used for contractor furnished procurements.
- Arrange for the contractor to make use of Government purchase agreements wherever there is a price advantage to do so. Often the contractor can get better price breaks than the Government.
- Include hardware and software procurement in the development contract.
- Understand your responsibility to ensure that GFE is provided on time and that it meets performance requirements.
- Assume GFE will be late and have a different interface than advertised.
- Check with your contracting officer for advice on existing purchase agreements, including the terms of those agreements.
- Establish minimum acceptability criteria for the contractor to procure equipment.

A.15 Multiple Development Contractors Used in Your Project

The Universal Truth: Avoid “Ping Pong” development.

A.15.1 Why This Is important

Do not allow your project to become a multi-contractor assembly line. An approach where each contractor adds a part as the software moves down a production line works well for cars, but not for software. Each set is essentially GFE from the last contractor to the next. This is bad because only the Government has any accountability and each contractor is successful whether the overall project works or not.

A.15.2 Benefits

- Concentrating software development within one organization will foster consistent development methodologies, tools, standards, and conventions.
- Employing a single development contractor will make subsequent integration and testing easier.

A.15.3 How Best To Emphasize

If splitting software development responsibilities among different organizations is unavoidable, ensure that the interfaces between distinct deliverables are simple and clearly defined:

- Deliverables should be testable, including interfaces, in a standalone mode prior to delivery for integration.
- Include cost and schedule to develop and deliver the necessary drivers to conduct such testing.
- Keep the skill base necessary to maintain and correct all code until final transition.
- Above all, avoid like the plague any situation that has modules being delivered and redelivered back and forth between contractors.
- Ensure someone besides the Government is fully responsible for each discrete program component.

Subcontracting is almost a given these days - get over it. Make subcontractor management a key clause in the contract and award fee structure. Specify that subs be treated as part of the team and not just as suppliers.

If you must employ multiple or geographically dispersed contractors identify this in your risk management plan.

A.16 Transitioning Everything to O&M

The Universal Truth: Deliver to the O&M organization all tools, drivers, and test data used during development

A.16.1 Why This Is important

It is certain that if you're developing contractor found test tools and test data necessary, operations and maintenance personnel will find them even more so. Operations and maintenance personnel are less familiar with the delivered code. Understanding and planning for this certainly will minimize delays, cost overruns, and unnecessary animosity between developers and maintainers.

A.16.2 Benefits

Your operations and maintenance organization will have the skills and tools it needs to be successful.

A.16.3 How Best To Emphasize

- Obtain agreement from the O&M organization regarding how relaxed the documentation standards can be for test tools, drivers, and data.
- Include in the RFP the requirement to deliver the test tools, drivers, and test data to the O&M organization.

A.17 Death March Project - When to Go for HELP!

The Universal Truth: Know when to hold-em, when to fold-em,
and when to RUN away!!!!!!!!
.

A.17.1 Why This Is important

Some programs, as conceived, are simply not executable. For whatever reason political, budget, hubris, etc. a well-intended project might become what could be termed a Death March project. These programs all have a set of common characteristics such as too much, too fast, too little resources, and/or too little trade space. Such programs break people and reputations.
Avoid them at all costs.

A.17.2 Benefits

Cost savings are recognized when Death March projects are killed early.

A.17.3 How Best To Emphasize

- Honestly assess what it takes for your project to succeed.
- If you have a Death March project in the making, renegotiate the underlying tenets of the project. If this is not possible, get an ironclad guarantee for a future, regardless of the outcome of the project, and a guaranteed medal if you succeed.
- Consider a career change. Maybe in hardware it's easier.

Appendix B Software Development Lifecycle

- Contract Award through SSR
  - System requirements analysis
  - System design
  - Software Requirements Analysis
- SSR Through PDR
  - Software Prelim Design
  - Preliminary Test Planning
    - Software implementation and unit testing
    - Unit integration and testing
  - PDR
- Post PDR Through CDR Activities
  - Detailed Design
  - Test Planning
  - CDR
- Post CDR Through TRR
  - Unit and Integration Testing
  - SWCI Qualification testing
  - SWCI/HWCI integration and testing
  - TRR
- System Qualification Testing
- Preparing for software use
- Preparing for software transition

Appendix C Checklists

The following checklists addresses a core set question that will define the capability to successfully perform the software development i.e., to meet the users needs. If you answer “no” to any of these questions you should examine the situation carefully for the possibility of greater risks to the project.

C.1 Requirements Engineering Checklist

This checklist is provided to assist the project officer in requirements engineering. If an item cannot be checked off as affirmative, then either rectify the situation or develop a contingency plan to solve problems that may arise.

Requirements Development:

- Have you had extensive user involvement in developing the requirements?
- Do all stakeholders understand and agree on how the system will be used?
- Are all stakeholders satisfied with the requirements?
- Do the developers understand the requirements?
- Are all requirements clear and unambiguous?
- Have you distinguished between needs and wants? Are requirements relevant?
- Are requirements consistent with each other (i.e., they don’t conflict.)?
- Are requirements complete? Do the requirements cover everything that is supposed to be accomplished?
- Has design detail been left out of the requirements?
- Are all requirements testable?
- Can you see the requirement as an output?
- Does each top-level software requirement have a unique identification number for tracking purposes?
- Are all requirements easily recognized, as requirements by using the word shall?
- Have the requirements been prioritized?
- Are requirements feasible with respect to cost, schedule, and technical capability?
- Are requirements verifiable?
- Has the contractor identified/performed any trade studies and risk reduction prototyping and demonstrations that should be accomplished as part of the iterative system requirements definition Process?
- For a spiral development, has the software level requirements for the current build/spiral been completely defined?
- For incremental development, has the software level requirements been identified for each increment at the beginning of the overall development effort?
- Has the contractor established and followed a systematic and disciplined process to ensure that the system and segment level requirements have been allocated to appropriate hardware and software configuration items?
- Are all external interfaces to the system clearly defined?
- Is the specification written so that it can be modified when necessary, with minimal impact to the rest of the document?

- Has the contractor accomplished an analysis to determine whether each allocated or derived software requirement can be met? Unachievable and/or unrealistic software requirements should be identified for resolution.
- Has the contractor accomplished a software requirements verifiability/testability/analysis to determine whether each requirement is objectively measurable? Any requirement that cannot be verified should be identified for resolution.
- Has the contractor identified all man/machine interfaces from both an operational and support standpoint? Once these are identified, the contractor should conduct human factors analyses as applicable to ensure that the design interface is consistent with sound human factors principles.
- Are you conducting formal and informal reviews of requirements documents?

Requirements Management:

- Have all requirements been entered into the requirements database?
- Does paragraph number, requirement number, or other useful index sort the requirements traces to allow requirements lookup?
- Can all requirements be traced to original system-level requirements?
- Are all system-level requirements allocated to lower level, subsystem requirements?
- Do you have a requirements change process documented and in place?
- Have you identified members of the requirements change board?
- Is adequate impact analysis performed for proposed requirements changes?
- Do you know who is responsible for making the changes?
- Have requirement changes been traced upward and downward through the higher and lower-level specifications?
- Do you have a process in place to maintain and control the different versions of the requirements specification?

C.2 Risk Management Checklist

This checklist is provided as to assist you in risk management. If you answer “no” to any of these questions you should examine the situation carefully for the possibility of greater risks to the project. This is only a cursory checklist for such an important subject. Please see the reference documents for more detailed checklists.

- Do you have a comprehensive, planned, and documented approach to risk management?
- Are all major areas/disciplines represented on your risk management team?
- Is the project manager experienced with similar projects?
- Do the stakeholders support disciplined development methods that incorporate adequate planning, requirements analysis, design, and testing?
- Is the project manager dedicated to this project, and not dividing his or her time among other efforts?
- Are you implementing a proven development methodology?
- Are requirements well defined, understandable, and stable?

- Do you have an effective requirements change process in place and do you use it?
- Does your project plan call for tracking/tracing requirements through all phases of the project?
- Are you implementing proven technology?
- Are suppliers stable, and do you have multiple sources for hardware and equipment?
- Are all procurement items needed for your development effort short-lead time items (no long-lead items?)
- Are all external and internal interfaces of the system well defined?
- Are all project positions appropriately staffed with qualified, motivated personnel?
- Are the developers trained and experienced in their respective development disciplines (i.e. systems engineering, software engineering, language, platform, tools, etc.?)
- Are developers experienced or familiar with the technology and the development environment?
- Are key personnel stable and likely to remain in their positions throughout the project?
- Is project funding stable and secure?
- Are all costs associated with the project known?
- Are development tools and equipment used for the project state-of-the-art, dependable, and available in sufficient quantity, and are the developers familiar with the development tools?
- Are the schedule estimates free of unknowns?
- Is the schedule realistic to support an acceptable level of risk?
- Is the project free of special environmental constraints or requirements?
- Is your testing approach feasible and appropriate for the components and system?
- Have acceptance criteria been established for all requirements and agreed to by all stakeholders?
- Will there be sufficient equipment to do adequate integration and testing?
- Has sufficient time been scheduled for system integration and testing?
- Can software be tested without complex testing or special test equipment?
- Is a single group in one location developing the system?
- Are subcontractors reliable and proven?
- Is all project work being done by groups over which you have control?
- Is development and support teams all collocated at one site?
- Is the project team accustomed to working on an effort of this size (neither bigger nor smaller?)?

C.3 Cost Management Checklist

This checklist is provided as to assist you in cost management. Consider your answers carefully to determine whether you need to examine the situation and take action.

- Is cost management planning part of your project planning process?
- Have you established a formal, documented cost management process?
- Do you have a complete and detailed WBS?
- Do you have historical information, including costs, from previous similar projects?
- Have you identified all sources of costs to your project (i.e. different types of labor, materials, supplies, equipment, etc.)?
- Have you identified proven and applicable estimating methods, models, and/or guides?
- Have you selected computer software to assist you in estimating, budgeting, tracking, and controlling costs?
- Do you have justifiable reasons for selecting your methods, models, guides, and software?
- Are cost issues adequately addressed in your risk management plan?
- Do you have a working change control process in place?
- Does the change control process adequately address cost impact?
- Do your estimates cover all tasks in the WBS?
- Do you understand your project's funding profile, i.e. how much funding will be provided? At what intervals? How sure is the funding?
- Have you developed a viable cost baseline that is synchronized with the project schedule and funding profile?
- Do you have adequate flexibility in the cost baseline?
- Do you have a plan/process for dealing with variances between cost performance and the baseline?
- Have you considered incorporating earned value management into your cost management efforts?
- Are you keeping records of your cost management activity for future efforts?

C.4 Measurement and Metrics Checklist

This checklist is provided to assist you in developing a metrics program, and defining and using metrics. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action. The checklist items are divided into three areas: developing, implementing, and reviewing a metrics program.

Developing a Metrics Program

- Do you have a planned and documented measurement/metrics and analysis plan?
- Are your metrics based on measurable or verifiable project goals?
- Do your project goals support the overall system-level goals?
- Are your goals well defined and unambiguous?
- Does each metric elicit only information that indicates progress toward or completion of a specific goal?
- Can questions on specific metrics be answered by providing specific information? (Is it unambiguous?)
- Does the metric provide all the information needed to determine progress or completion of the goal?
- Is each metric required for specific decision-making activities?
- Is each metric derived from two or more measurements (e.g. Remaining budget vs. schedule)?
- Have you documented the analysis methods used to calculate the metrics?
- Have you defined those measures needed to provide the metrics?
- Have you defined the collection process (i.e. what, how, who, when, how often, etc.)?

Metrics Program Implementation:

- Does your implementation follow the metrics program plan?
- Is data collected the same way each time it is collected?
- Are documented analysis methods followed when calculating metrics?
- Are metrics delivered in a timely manner to those who need them?
- Are metrics being used in the decision making process?
- Metrics Program Evaluation
- Are the metrics sufficient?
- Are all metrics or measures required, that is, non-superfluous?
- Are measurements allowing project work to continue without interference?
- Does the analysis produce accurate results?
- Is the data collection interval appropriate?
- Is the metrics program as simple as it can be while remaining adequate?
- Has the metrics program been modified to adequately accommodate any project or organizational goal changes?

C.5 Configuration Management Checklist

This checklist is provided to assist you in establishing an effective CM program. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action.

CM Planning:

- Have you planned and documented a configuration management process?
- Have you identified Configuration Control Board members for each needed control board?
- Has CM software been chosen to facilitate your CM process?

Establishing Baselines:

- Have all configuration items been identified?
- Have baselines been established for all configuration items?
- Has a descriptive schema been developed to accurately identify configuration items and changes to their configuration?

Controlling, Documenting, And Auditing:

- Is there a formal process for documenting and submitting proposed changes?
- Is the Configuration Control Board active and responsible in evaluating and approving changes?
- Is there a "higher authority" to appeal to when the CCB gets "hung," and can't come to a consensus?
- Are all changes tracked until they are fully implemented?
- Are all changes fully documented in the baseline documents and change histories?
- Are regular reports and configuration updates published and distributed to interested organizations?
- Are regular audits and reviews performed to evaluate configuration integrity?
- Are configuration errors dealt with in an efficient and timely manner?

Updating the Process:

- Is the CM program itself – its efficiency, responsiveness, and accuracy – evaluated regularly?
- Is the CM program modified to include recommended improvements when needed?

C.6 Software Engineering Processes Checklist

This checklist is provided to assist you in understanding the software engineering issues of your project. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action.

Before Starting:

- Do you know what software development life cycle your project will be employing and how it coordinates with the software and project life cycles?
- Does the development team have experience in the software development life cycle to be used?

- Do the developers, the stakeholders, and you understand what the steps of the development process are, and what the inputs and products of each step are?
- Has your project been planned in the various software development areas?
- Do you know what design method has been chosen for the development effort and why it was chosen over other methods?
- Does your development team have experience with the chosen design method? If not, has the schedule been adjusted to allow for learning the new design method?
- Have proven CASE tools been chosen to assist in the software design?
- Does your development team have experience with the chosen CASE tools? If not, has the schedule been adjusted to allow for learning to use the CASE tools?
- Has an appropriate programming language been chosen and do you know the reasons it was chosen?
- Does your development team have experience with the chosen programming language? If not, has the schedule been adjusted to allow for learning the chosen programming language?
- Is the development team sufficiently skilled and experienced in programming to properly and efficiently design, code, and test the software?

During Project Execution:

- Are your requirements complete, unambiguous, and agreed to by both developers and stakeholders?
- Have you completed both system and functional specifications, and have they been reviewed and approved by stakeholders?
- Is the development team familiar with or provided with the appropriate opportunity to become familiar with the operating system and system hardware?
- Is a detailed software design being completed, reviewed, and approved before coding starts?
- Is testing being properly implemented and satisfactorily completed at unit, integration, and system levels before acceptance testing?
- Are human factors being considered sufficiently in the software design?

At Completion:

- Does the completed software correctly implement the design?
- Does the software meet the requirements?
- Does the software meet the users' needs?

C.7 Testing Checklist

This checklist is provided to assist you in understanding the testing issues of your project. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action.

Before Starting:

- Is testing planned for and considered throughout the entire development life cycle?
- Is the overall testing strategy defined and documented, and is it an integral part of and consistent with the development program?

- Is the testing process well defined, documented, understood, and supported by the development team and management?
- Are test requirements clearly defined?
- Are test methods, techniques, controls, and standards clearly defined and consistent with the testing strategy?
- Is each test activity traceable to specific requirements?
- Are configuration management and quality assurance in place and are they adequate to support the testing strategy?
- Are testers trained, skilled, and motivated people?
- Have adequate time and resources been reserved for testing?
- Are time and resources allocated for test preparation early in the project life cycle?

During Execution:

- Is testing used as a primary tool to ensure good project health?
- Is testing implemented as a tool for improving product quality and the development process as a whole?
- Is early life cycle testing used to prevent propagation of defects to later stages of development?
- Is a tracking system being used to record what has been tested and what has not?
- Is a database of test results being maintained for current and future reference?
- Are tests used as milestone and progress indicators?
- Is the right amount of testing being done to balance risk with available time and resources?
- Are you using inspections and other evaluation methods (see Chapter 11) to reduce the errors found through testing?
- Do you know when your testing is complete?

C.8 Sustainment Checklist

This checklist is provided to assist you in understanding the sustainment and product improvement issues of your project. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action.

Sustainment:

- Is all your software developed with a goal to facilitate its future sustainment?
- Do you understand the place and purpose of the sustainment phase in the software life cycle?
- Do you understand your sustainment process?
- Is there a sustainment plan?
- Is there a process in place to gather problem reports and upgrade requests for the software?
- Does the plan provide for reviewing, evaluating, and prioritizing upgrade requests?
- Are all sustainment activity steps included in the plan?
- Is there a transition plan to move to the upgraded system?

- Have all activities been planned and organized to keep interference and downtime to the operating system to a minimum?
- Does the plan call for running critical systems redundantly during testing and installation?
- Do the deliveries include source code, documentation, and all else that is needed in addition to the software itself to continue maintaining the software?
- Are all products under configuration control?

C.9 Formal Reviews Checklist

Formal reviews may include reviewing SQA, SV&V, and SCM results in order to examine the product. This review may also help detect whether or not these activities were performed in accordance with their respective plans. The checklists below provide a guideline for reviewing both the product and plans for assurance activities. These checklists are not necessarily complete; at any time the reviewer during development may need to expand upon a given topic. In all checklists, negative answers require further examination by the reviewers.

C.9.1 Software Requirements Review

The following checklist contains questions a reviewer during development to ask at the software requirements review based on:

Compatibility:

- Do the interface requirements enable compatibility of external interfaces (hardware and software)?

Completeness:

- Does the SRS contain everything listed in the corresponding documentation content? Does it include all requirements relating to functionality, performance, constraints, safety, etc.?
- Does SRS include all user requirements (as defined in the concept phase)?
- Do the functional requirements cover all abnormal situations?
- Have the temporal aspects of all functions been considered?
- Are the time-critical functions identified and the time criteria for them specified? Do these include the maximum and minimum times for their execution?
- Does SRS define those requirements for which future changes are anticipated?
- Are all normal environmental variables included?
- Are the environmental conditions specified for all operating modes (e.g., normal, abnormal, disturbed)?

Consistency:

- Is there internal consistency between the software requirements?
- Is the SRS free of contradictions?
- Are the specified models, algorithms, and numerical techniques compatible?
- Does SRS use standard terminology and definitions throughout?
- Is SRS compatible with the operational environment of the hardware and software?

- Has the impact of software on the system and environment been specified?
- Has the impact of the environment on the software been specified?

Correctness:

- Does the SRS conform to SRS standards?
- Are algorithms and regulations supported by scientific or other appropriate literature?
- What evidence is there that shows vendor has applied the regulations correctly?
- Does the SRS define the required responses to all expected types of errors and failure modes identified by the hazard analysis?
- Were the functional requirements analyzed to check if all abnormal situations are covered by system functions?
- Does SRS reference desired development standards?
- Does the SRS identify external interfaces in terms of input and output mathematical variables?
- Are the requirements for the man-machine interface adequate?
- What is the rationale for each requirement? Is it adequate?
- Is there justification for the design/implementation constraints?

Feasibility:

- Are the design, operation, and maintenance of software feasible?
- Are the specified models, numerical techniques, and algorithms appropriate for the problem to be solved? Are they accepted practice for nuclear power plants? Can they be implemented within the imposed constraints?
- Are the quality attributes specified achievable individually and as a group?

Modifiability:

- Are requirements organized so as to allow for modifications (e.g., with adequate structure and cross referencing)?
- Is each unique requirement defined more than once? Are there any redundant statements?
- Is there a set of rules for maintaining the SRS for the rest of the software lifecycle?

Robustness

- Are there requirements for fault tolerance and graceful degradation?

Traceability:

- Is there traceability from the next higher-level spec (e.g., system concept/requirements and user needs as defined in concept phase, and system design)?
- Does the SRS show explicitly the mapping and complete coverage of all relevant requirements and design constraints defined in the concept phase, by such means as a coverage matrix or cross-reference?
- Is SRS traceable forward through successive development phases (e.g., into the design, code, and test documentation)?
- Are safety functions or computer security functions flagged?

Understandability:

- Does every requirement have only one interpretation?
- Are the functional requirements in modular form with each function explicitly identified?
- Is there a glossary of terms?
- Is formal or semiformal language used?
- Is the language ambiguous?
- Does the SRS contain only necessary implementation details and no unnecessary details? Is it over specified?
- Are the requirements clear and specific enough to be the basis for detailed design specs and functional test cases?
- Does the SRS differentiate between program requirements and other information provided?

Verifiability/Testability:

- Are the requirements verifiable (i.e., can the software be checked to see whether requirements have been fulfilled)?
- Are mathematical functions defined using notation with precisely defined syntax and semantics?
- Is there a verification procedure defined for each requirement in the SRS?

C.9.2 Software Design Review

Reviewers should be able to determine whether or not all design features are consistent with the requirements. Numerical techniques and algorithms should be appropriate for the problem to be solved. The program design needs to be partitioned in a manner consistent with the problem to be solved. And, the program should meet the requirements. The following checklist contains questions a reviewer during development may ask at the software design review.

Completeness:

- Are the SRS requirements fulfilled?
- Is there enough data (logic diagrams, algorithms, storage allocation charts, etc.) available to ensure design integrity?
- Are algorithms and equations adequate, accurate, and complete?
- Are requirements for the support and test software and hardware to be used in the development of the product included?
- Does the design implement required program behavior with respect to each program interface?
- Are all program inputs, outputs, and database elements identified and described to the extent needed to code the program?
- Does the SDD describe the operational environment into which the program must fit?
- Are all required processing steps included?
- Are all possible outcomes of each decision point designated?
- Does the design take into account all expected situations and conditions?

- Does the design specify appropriate behavior in the face of unexpected or improper inputs and other anomalous conditions?
- Does the SDD reference all desired programming standards?

Consistency:

- Are standard terminology and definitions used throughout the SDD? Are the style of presentation and the level of detail consistent throughout the document?
- Does the design configuration ensure integrity of changes?
- Is there compatibility of the interfaces?
- Is the test documentation compatible with the test requirements of the SRS?
- Is the SDD free of internal contradictions?
- Are the models, algorithms, and numerical techniques that are specified mathematically compatible?
- Is input and output formats consistent to the extent possible?
- Are the designs for similar or related functions consistent?
- Are the accuracies and units of inputs, database elements, and outputs that are used together in computations or logical decisions compatible?

Correctness:

- Does the SDD conform to design documentation standards?
- Does the design perform only that which is specified in the SRS unless additional functionality is justified?
- Is the test documentation current and technically accurate?
- Is the design logic sound -- will the program do what is intended?
- Is the design consistent with documented descriptions and know properties of the operational environment into which the program must fit?
- Do interface designs agree with documented descriptions and known properties of the interfacing elements?
- Does the design correctly accommodate all inputs, outputs, and database elements whose format, content, data rate, etc. are not at the discretion of the designer?

Feasibility:

- Are the specified models, algorithms, and numerical techniques accepted practices for use on Space Systems?
- Can they be implemented within the constraints imposed on the system and on the development effort?
- Are the functions as designed implement able within the available resources?

Modifiability:

- Does the design use information hiding as follows:
  - The modules are organized such that changes in the requirements only require changes to a small number of modules.
  - Functions and data structures likely to changes have interfaces insensitive to changes in individual functions.
  - The design partitions data structure access, database access and I/O access from the application software by the use of access programs (globally accessible data is not used).

    - Functionality is partitioned into programs to maximize the internal cohesion of programs and to minimize program coupling.
- Does each program have a single function?

Modularity:

- Is there a schema for modularity, e.g., model-based?
- Is the design structured so that it comprises relatively small, hierarchically related programs or sets of programs, each performing a particular, unique function?
- Do the design use specific criteria to limit program size?

Predictability:

- Does the design contain programs that provide the required response to identified error conditions?
- Does the design, schedule computer resources in a manner that is primarily deterministic and predictable rather than dynamic?
- Does the design contain a minimum number of interrupts and event driven software? Is justification given for uses of these features?
- Is plausibility checking performed on the execution of programs to uncover errors associated with the frequency and/or order or program execution and the permissiveness of program execution?

Robustness:

- Are all SRS requirements related to fault tolerance and graceful degradation addressed in the design?

Structured-ness:

- Does the design use a logical hierarchical control structure?

Traceability:

- Does the SDD show mapping and complete coverage of all requirements and design constraints in the SRS?
- Are all functions in the SDD outside the scope of the SRS identified?
- Are all functions identified so they can be uniquely reference by the code?
- Does the SDD contain or reference a revision history that identifies all modifications to the design and the rationale for these changes?
- Does the SDD reference the design notes that document design decisions relevant to the software design?
- Have safety and computer security functions been flagged?

Understandability:

- Does the SDD avoid unnecessarily complex designs and design representations.
- Is the SDD written to allow unambiguous interpretation?
- Verifiability/Testability
- Does the SDD describe each function using well-defined notation so that the SDD can be verified against the SRS and the code can be verified against the SDD?
- Are conditions, constraints identified quantitatively so that tests may be designed?

C.9.3 Source Code Review

The following checklist contains the kinds of questions a reviewer during development may ask at the source code review.

Completeness:

- Is the code a complete and precise implementation of the design as documented in the SDD?
- Was the code integrated and debugged to satisfy the design specified in the SDD?
- Does the code create the required databases, including the appropriate initial data?
- Are there any un-referenced or undefined variables, constants, or data types?

Consistency:

- Is the code logically consistent with the SDD?
- Are the same format, invocation convention, and structure used throughout?

Correctness:

- Does the code conform to specified standards?
- Are all variables properly specified and used?
- Are all comments accurate?
- Are all programs invoked with the correct number of parameters?

Modifiability:

- Does the code refer to constants symbolically to facilitate change?
- Are cross-references or data dictionaries included to show variable and constant access by the program?
- Does code consist of programs with only one entry point and one exit point (exception is with fatal error handling)?
- Does code reference labels or other symbolic constants rather than addresses?

Predictability:

- Is the code written in a language with well-defined syntax and semantics:
- Was the use of self-modifying code avoided?
- Does the code avoid relying on defaults provided by the programming language?
- Is the code free of unintended infinite loops?
- Does the code avoid recursion?

Robustness:

- Does the code protect against detectable runtime errors (e.g., range array index values, division by zero, out of range variable values, and stack overflow)?

Structured-ness:

- Is each function of the program recognizable as a block of code?
- Do loops only have one entrance?

Traceability:

- Does the code identify each program uniquely?
- Is there a cross-reference framework through which the code can be easily and directly traced to the SRS, SDD, and Test Cases/Procedures?
- Does the code contain or reference a revision history of all code modifications and the reason for them?
- Have all safety and computer security functions been flagged?

Understandability:

- Do the comment statements adequately describe each routine, using clear English language?
- Were ambiguous or unnecessarily complex coding used? If so, are they clearly commented?
- Were consistent formatting techniques (e.g., indentation, use of white space) used to enhance clarity?
- Was a mnemonic naming convention used? Does the naming reflect the type of variable?
- Is the valid range of each variable defined?
- Does the code use mathematical equations that correspond to the mathematical models described/derived in the SDD?

Verifiability:

- Are implementation practices and techniques that are difficult to test avoided?

C.9.4 Test Readiness Review

The test readiness review is usually conducted following completion of component testing or software integration testing. The purpose is to ensure readiness to begin formal integration testing or system testing without complications, and to ensure that test documentation is complete, that errors have been removed, and that use of the test facilities has been planned properly. The following are general questions that might be asked at these reviews:

- Is the code under configuration control?
- Have recommended changes been made to the code as result of source code review or, as appropriate, component test or integration test?
- Is the error rate sufficiently low to warrant beginning the next type of testing?
- Are all test cases/procedures complete?
- Is the test facility ready? Are schedules approved, personnel and physical requirements specified?
- Have all test tools been checked?
- Have all test procedures been checked?

C.9.5 Test Report Review

The purpose of this review is to assure the completeness of test activities. It is usually required at the end of development (i.e., following system testing). However, it may be conducted after completion of module, integration, or system testing.

The reviewer should ask questions about whether the test objectives were met (e.g., whether all test cases were executed, whether test results matched the expected outputs, and whether the results were traceable to the test cases and software requirements).

The reviewer should also inquire about the nature of major anomalies (e.g., whether there were similarities, what corrective actions were taken). The reviewer should check that the anomalies existed only in the product, and were not caused by the test cases and procedures.

C.9.6 Development Documentation Review

A development documentation review should be conducted following completion of the testing phase, to "assure completion and acceptability of the development documentation." It is assumed that this consists of all documentation produced during the development phases of the lifecycle (e.g., requirements, design, coding). The purpose of the development documentation review is to check for consistency among the development documents and to ensure that all changes have been made consistently.

C.9.7 Installation and Checkout Review

The following checklist contains the kinds of questions a reviewer during development may ask at the installation and checkout review.

Completeness:

- Are elements necessary for rebuilding and testing of the installed program available on the installation medium (e.g., source code, user-supplied library routines, and test cases)?
- Has the vendor provided adequate information for installing the program in more than one operating environment?
-

Correctness:

- Can all test cases be performed?
- Do the test cases produce results identical to the expected outputs?
- Are all results identical to previous results (when the same tests are executed more than once)? If not, are differences in results clearly understood and justified?

Understandability:

- Are the format and content of the medium properly identified for easy reading of the files?
- Are the installation procedures clearly understandable?

C.10 COTS Checklist

This checklist is provided to assist you in understanding of COTS issues of your project. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action.

- Do you conduct make vs. buy vs. rent trade studies instead of just assuming that buy is the right choice?
- Do you use your requirements as the criteria for your trade studies?
- Do you consider the full life cycle when deciding whether to make, buy, or rent?
- Do you know all your options?
- Are you knowledgeable of the marketplace and the vendors?
- Does the vendor understand your needs?
- Are you not lowering your requirements indiscriminately to use COTS?
- Do you understand the total life cycle costs?
- Are the product and vendor likely to be around for the lifetime of the system?
- Have you satisfactorily resolved all security issues?
- Have you reduced all known risks to an acceptable level?

Appendix D Software Quality Attributes

Compatibility:

- The ability of two or more systems or components to perform their required functions while sharing the same hardware or software environment.
- The ability of two or more systems or components to exchange information.

Completeness:

- The degree to which all of the software's required functions and design constraints are present and fully developed in the SRS, SDD, and code.

Consistency:

- The degree of uniformity, standardization, and freedom from contradiction among the documents or parts of a system or component. See also TRACEABILITY.

Correctness:

- The degree to which a system or component is free from faults in its specification, design, and implementation.
- The degree to which software, documentation, or other items meet specified requirements.
- The degree to which software, documentation, or other items meet user needs and expectations, whether specified or not.
- The ability of the SRS, SDD, and code to describe or produce the specified outputs when given the specified inputs, and the extent to which they match or satisfy the requirements.

Feasibility:

- The degree to which the requirements, design, or plans for a system or component can be implemented under existing constraints.
- Modifiability Quality attribute refers to the characteristics of the SRS, SDD, and code that facilitate the incorporation of changes.

Modularity:

- The degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.

Predictability:

- The degree to which the functionality and performance of the software are deterministic for a specified set of inputs.

Robustness:

- The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions.

Structured-ness:

- The degree to which the SDD and code possess a definite pattern in their interdependent parts. This implies that the design has proceeded in an orderly and systematic manner (e.g., top-down), has minimized coupling between modules, and that standards control structures have been followed during coding resulting in well structured software.

Testability:

- The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met.
- The degree to which a requirement is stated in terms that permits establishment of test criteria and performance of tests to determine whether those criteria have been met.

Traceability:

- The degree to which a relationship can be established between two or more products of the development process.
- The degree to which each element in a software development product establishes its reason for existing (e.g., the degree to which each element in a bubble chart references the requirement that it satisfies).
- The ability to trace the design decision history and reasons for these changes. See also CONSISTENCY.

Understandability:

- The degree to which the meaning of the SRS, SDD, and code are clear to the reader.

Verifiability:

- The degree to which the SRS, SDD, and code have been written to facilitate verification using both static methods and testing.

Appendix E Software Development Plan DID DI-MCCR-80030A

1. SOFTWARE DEVELOPMENT PLAN (SDP)
2. IDENTIFICATION NUMBER
DI-IPSC-81427
3. DESCRIPTION/PURPOSE
3.1 The Software Development Plan (SDP) describes a developer's plans for conducting a software development effort. The term "software development" in this DID is meant to include new development, modification, reuse, reengineering, maintenance, and all other activities resulting in software products.
3.2 The SDP provides the acquirer insight into, and a tool for monitoring, the processes to be followed for software development, the methods to be used, the approach to be followed for each activity, and project schedules, organization, and resources.
4. APPROVAL DATE (YYMMDD)
941205
5. OFFICE OF PRIMARY RESPONSIBILITY
EC
6a. DTIC APPLICABLE
6b. GIDEP APPLICABLE
7. APPLICATION/INTERRELATIONSHIP
7.1 This Data Item Description (DID) contains the format and content preparation instructions for the data product generated by specific and discrete task requirements as delineated in the contract.
7.2 This DID is used when the developer is tasked to develop and record plans for conducting software development activities.
7.3 Portions of this plan may be bound separately if this approach enhances their usability. Examples include plans for software configuration management and software quality assurance.
7.4 The Contract Data Requirements List (CDRL) (DD 1423) should specify whether deliverable data are to be delivered on paper or electronic media; are to be in a given electronic form (such as ASCII, CALS, or compatible with a specified word processor or other support software); may be delivered in developer format rather than in the format specified herein; and may reside in a computer-aided software engineering (CASE) or other automated tool rather than in the form of a traditional document.
7.5 This DID supersedes DI-MCCR-80030A, DI-MCCR-80297, DI-MCCR-80298, DI-MCCR-80299, DI-MCCR-80300, and DI-MCCR-80319.
8. APPROVAL LIMITATION
Limited Approval from 12/5/94 through 12/5/96
9a. APPLICABLE FORMS
9b. AMSC NUMBER
N7070
10. PREPARATION INSTRUCTIONS
10.1 General instructions.
a. Automated techniques. Use of automated techniques is encouraged. The term "document" in this DID means a collection of data regardless of its medium.

b. Alternate presentation styles. Diagrams, tables, matrices, and other presentation styles are acceptable substitutes for text when data required by this DID can be made more readable using these styles.
c. Title page or identifier. The document shall include a title page containing, as applicable: document number; volume number; version/revision indicator; security markings or other restrictions on the handling of the document; date; document title; name, abbreviation, and any other identifier for the system, subsystem, or item to which the document applies; contract number; CDRL item number; organization for which the document has been prepared; name and address of the preparing organization; and distribution statement. For data in a database or other alternative form, this information shall be included on external and internal labels or by equivalent identification methods.
d. Table of contents. The document shall contain a table of contents providing the number, title, and page number of each titled paragraph, figure, table, and appendix. For data in a database or other alternative form, this information shall consist of an internal or external table of contents containing pointers to, or instructions for accessing, each paragraph, figure, table, and appendix or their equivalents.
e. Page numbering/labeling. Each page shall contain a unique page number and display the document number, including version, volume, and date, as applicable. For data in a database or other alternative form, files, screens, or other entities shall be assigned names or numbers in such a way that desired data can be indexed and accessed.
f. Response to tailoring instructions. If a paragraph is tailored out of this DID, the resulting document shall contain the corresponding paragraph number and title, followed by "This paragraph has been tailored out." For data in a database or other alternative form, this representation need occur only in the table of contents or equivalent.
g. Multiple paragraphs and subparagraphs. Any section, paragraph, or subparagraph in this DID may be written as multiple paragraphs or subparagraphs to enhance readability.
h. Standard data descriptions. If a data description required by this DID has been published in a standard data element dictionary specified in the contract, reference to an entry in that dictionary is preferred over including the description itself.
i. Substitution of existing documents. Commercial or other existing documents, including other project plans, may be substituted for all or part of the document if they contain the required data
10.2 Content requirements.
Content requirements begin on the following page. The numbers showed designate the paragraph numbers to be used in the document. Each such number is understood to have the prefix "10.2" within this DID. For example, the paragraph numbered 1.1 is understood to be paragraph 10.2.1.1 within this DID.
1. Scope
This section shall be divided into the following paragraphs.
1.1 Identification.
This paragraph shall contain a full identification of the system and the software to which this document applies, including, as applicable, identification number(s), title(s), abbreviation(s), version number(s), and release number(s).

1.2 System overview.
This paragraph shall briefly state the purpose of the system and the software to which this document applies. It shall describe the general nature of the system and software; summarize the history of system development, operation, and maintenance; identify the project sponsor, acquirer, user, developer, and support agencies; identify current and planned operating sites; and list other relevant documents.
1.3 Document overview.
This paragraph shall summarize the purpose and contents of this document and shall describe any security or privacy considerations associated with its use.
1.4 Relationship to other plans.
This paragraph shall describe the relationship, if any, of the SDP to other project management plans.
2. Referenced documents.
This section shall list the number, title, revision, and date of all documents referenced in this plan. This section shall also identify the source for all documents not available through normal Government stocking activities.
3. Overview of required work.
This section shall be divided into paragraphs as needed to establish the context for the planning described in later sections. It shall include, as applicable, an overview of:
a. Requirements and constraints on the system and software to be developed
b. Requirements and constraints on project documentation
c. Position of the project in the system life cycle
d. The selected program/acquisition strategy or any requirements or constraints on it
e. Requirements and constraints on project schedules and resources
f. Other requirements and constraints, such as on project security, privacy, methods, standards, interdependencies in hardware and software development, etc.
4. Plans for performing general software development activities.
This section shall be divided into the following paragraphs. Provisions corresponding to non-required activities may be satisfied by the words "Not applicable." If different builds or different software on the project require different planning, these differences shall be noted in the paragraphs. In addition to the content specified below, each paragraph shall identify applicable risks/uncertainties and plans for dealing with them.
4.1 Software development process.
This paragraph shall describe the software development process to be used. The planning shall cover all contractual clauses concerning this topic, identifying planned builds, if applicable, their objectives, and the software development activities to be performed in each build.
4.2 General plans for software development.
This paragraph shall be divided into the following subparagraphs.
4.2.1 Software development methods.
This paragraph shall describe or reference the software development methods to be used. Included shall be descriptions of the manual and automated tools and procedures to be used in support of these methods. The methods shall cover all contractual clauses concerning this topic. Reference may be made to other paragraphs in this plan if the methods are better described in context with the activities to which they will be applied.

4.2.2 Standards for software products.
This paragraph shall describe or reference the standards to be followed for representing requirements, design, code, test cases, test procedures, and test results. The standards shall cover all contractual clauses concerning this topic. Reference may be made to other paragraphs in this plan if the standards are better described in context with the activities to which they will be applied. Standards for code shall be provided for each programming language to be used. They shall include at a minimum:
a. Standards for format (such as indentation, spacing, capitalization, and order of information)
b. Standards for header comments (requiring, for example, name/identifier of the code; version identification; modification history; purpose; requirements and design decisions implemented; notes on the processing (such as algorithms used, assumptions, constraints, limitations, and side effects); and notes on the data (inputs, outputs, variables, data structures, etc.)
c. Standards for other comments (such as required number and content expectations)
d. Naming conventions for variables, parameters, packages, procedures, files, etc.
e. Restrictions, if any, on the use of programming language constructs or features
f. Restrictions, if any, on the complexity of code aggregates
4.2.3 Reusable software products.
This paragraph shall be divided into the following subparagraphs.
4.2.3.1 Incorporating reusable software products.
This paragraph shall describe the approach to be followed for identifying, evaluating, and incorporating reusable software products, including the scope of the search for such products and the criteria to be used for their evaluation. It shall cover all contractual clauses concerning this topic. Candidate or selected reusable software products known at the time this plan is prepared or updated shall be identified and described, together with benefits, drawbacks, and restrictions, as applicable, associated with their use.
4.2.3.2 Developing reusable software products.
This paragraph shall describe the approach to be followed for identifying, evaluating, and reporting opportunities for developing reusable software products. It shall cover all contractual clauses concerning this topic.
4.2.4 Handling of critical requirements.
This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for handling requirements designated critical. The planning in each subparagraph shall cover all contractual clauses concerning the identified topic.
4.2.4.1 Safety assurance
4.2.4.2 Security assurance
4.2.4.3 Privacy assurance
4.2.4.4 Assurance of other critical requirements
4.2.5 Computer hardware resource utilization.
This paragraph shall describe the approach to be followed for allocating computer hardware resources and monitoring their utilization. It shall cover all contractual clauses concerning this topic.

4.2.6 Recording rationale.
This paragraph shall describe the approach to be followed for recording rationale that will be useful to the support agency for key decisions made on the project. It shall interpret the term "key decisions" for the project and state where the rationale are to be recorded. It shall cover all contractual clauses concerning this topic.
4.2.7 Access for acquirer review.
This paragraph shall describe the approach to be followed for providing the acquirer or its authorized representative access to developer and subcontractor facilities for review of software products and activities. It shall cover all contractual clauses concerning this topic.
5. Plans for performing detailed software development activities.
This section shall be divided into the following paragraphs. Provisions corresponding to non-required activities may be satisfied by the words "Not applicable." If different builds or different software on the project require different planning, these differences shall be noted in the paragraphs. The discussion of each activity shall include the approach (methods/procedures/tools) to be applied to:
1) the analysis or other technical tasks involved,
2) the recording of results, and
3) the preparation of associated deliverables, if applicable. The discussion shall also identify applicable risks/uncertainties and plans for dealing with them. Reference may be made to 4.2.1 if applicable methods are described there.
5.1 Project planning and oversight.
This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for project planning and oversight. The planning in each subparagraph shall cover all contractual clauses regarding the identified topic.
5.1.1 Software development planning (covering updates to this plan)
5.1.2 CSCI test planning
5.1.3 System test planning
5.1.4 Software installation planning
5.1.5 Software transition planning
5.1.6 Following and updating plans, including the intervals for management review
5.2 Establishing a software development environment.
This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for establishing, controlling, and maintaining a software development environment. The planning in each subparagraph shall cover all contractual clauses regarding the identified topic.
5.2.1 Software engineering environment
5.2.2 Software test environment
5.2.3 Software development library
5.2.4 Software development files
5.2.5 Non-deliverable software
5.3 System requirements analysis.
This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for participating in system requirements analysis. The planning in each subparagraph shall cover all contractual clauses regarding the identified topic.
5.3.1 Analysis of user input

5.3.2 Operational concept
5.3.3 System requirements
5.4 System design.
This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for participating in system design. The planning in each subparagraph shall cover all contractual clauses regarding the identified topic.
5.4.1 System-wide design decisions
5.4.2 System architectural design
5.5 Software requirements analysis.
This paragraph shall describe the approach to be followed for software requirements analysis. The approach shall cover all contractual clauses concerning this topic.
5.6 Software design.
This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for software design. The planning in each subparagraph shall cover all contractual clauses regarding the identified topic.
5.6.1 CSCI-wide design decisions
5.6.2 CSCI architectural design
5.6.3 CSCI detailed design
5.7 Software implementation and unit testing.
This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for software implementation and unit testing. The planning in each subparagraph shall cover all contractual clauses regarding the identified topic.
5.7.1 Software implementation
5.7.2 Preparing for unit testing
5.7.3 Performing unit testing
5.7.4 Revision and retesting
5.7.5 Analyzing and recording unit test results
5.8 Unit integration and testing.
This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for unit integration and testing. The planning in each subparagraph shall cover all contractual clauses regarding the identified topic.
5.8.1 Preparing for unit integration and testing
5.8.2 Performing unit integration and testing
5.8.3 Revision and retesting
5.8.4 Analyzing and recording unit integration and test results
5.9 CSCI qualification testing.
This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for CSCI qualification testing. The planning in each subparagraph shall cover all contractual clauses regarding the identified topic.
5.9.1 Independence in CSCI qualification testing
5.9.2 Testing on the target computer system
5.9.3 Preparing for CSCI qualification testing
5.9.4 Dry run of CSCI qualification testing
5.9.5 Performing CSCI qualification testing
5.9.6 Revision and retesting
5.9.7 Analyzing and recording CSCI qualification test results

5.10 CSCI/HWCI integration and testing.
This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for participating in CSCI/HWCI integration and testing. The planning in each subparagraph shall cover all contractual clauses regarding the identified topic.
5.10.1 Preparing for CSCI/HWCI integration and testing
5.10.2 Performing CSCI/HWCI integration and testing
5.10.3 Revision and retesting
5.10.4 Analyzing and recording CSCI/HWCI integration and test results
5.11 System qualification testing.
This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for participating in system qualification testing. The planning in each subparagraph shall cover all contractual clauses regarding the identified topic.
5.11.1 Independence in system qualification testing
5.11.2 Testing on the target computer system
5.11.3 Preparing for system qualification testing
5.11.4 Dry run of system qualification testing
5.11.5 Performing system qualification testing
5.11.6 Revision and retesting
5.11.7 Analyzing and recording system qualification test results
5.12 Preparing for software use.
This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for preparing for software use. The planning in each subparagraph shall cover all contractual clauses regarding the identified topic.
5.12.1 Preparing the executable software
5.12.2 Preparing version descriptions for user sites
5.12.3 Preparing user manuals
5.12.4 Installation at user sites
5.13 Preparing for software transition.
This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for preparing for software transition. The planning in each subparagraph shall cover all contractual clauses regarding the identified topic.
5.13.1 Preparing the executable software
5.13.2 Preparing source files
5.13.3 Preparing version descriptions for the support site
5.13.4 Preparing the "as built" CSCI design and other software support information
5.13.5 Updating the system design description
5.13.6 Preparing support manuals
5.13.7 Transition to the designated support site
5.14 Software configuration management.
This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for software configuration management. The planning in each subparagraph shall cover all contractual clauses regarding the identified topic.
5.14.1 Configuration identification
5.14.2 Configuration control
5.14.3 Configuration status accounting

5.14.4 Configuration audits
5.14.5 Packaging, storage, handling, and delivery
5.15 Software product evaluation.
This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for software product evaluation. The planning in each subparagraph shall cover all contractual clauses regarding the identified topic.
5.15.1 In-process and final software product evaluations
5.15.2 Software product evaluation records, including items to be recorded
5.15.3 Independence in software product evaluation
5.16 Software quality assurance.
This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for software quality assurance. The planning in each subparagraph shall cover all contractual clauses regarding the identified topic.
5.16.1 Software quality assurance evaluations
5.16.2 Software quality assurance records, including items to be recorded
5.16.3 Independence in software quality assurance
5.17 Corrective action.
This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for corrective action. The planning in each subparagraph shall cover all contractual clauses regarding the identified topic.
5.17.1 Problem/change reports,
including items to be recorded (candidate items include project name, originator, problem number, problem name, software element or document affected, origination date, category and priority, description, analyst assigned to the problem, date assigned, date completed, analysis time, recommended solution, impacts, problem status, approval of solution, follow-up actions, corrector, correction date, version where corrected, correction time, description of solution implemented)
5.17.2 Corrective action system
5.18 Joint technical and management reviews.
This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for joint technical and management reviews. The planning in each subparagraph shall cover all contractual clauses regarding the identified topic.
5.18.1 Joint technical reviews, including a proposed set of reviews
5.18.2 Joint management reviews, including a proposed set of reviews
5.19 Other software development activities.
This paragraph shall be divided into the following subparagraphs to describe the approach to be followed for other software development activities. The planning in each subparagraph shall cover all contractual clauses regarding the identified topic.
5.19.1 Risk management, including known risks and corresponding strategies
5.19.2 Software management indicators, including indicators to be used
5.19.3 Security and privacy
5.19.4 Subcontractor management
5.19.5 Interface with software independent verification and validation (IV&V) agents
5.19.6 Coordination with associate developers
5.19.7 Improvement of project processes
5.19.8 Other activities not covered elsewhere in the plan

6. Schedules and activity network.
This section shall present:
a. Schedule(s) identifying the activities in each build and showing initiation of each activity, availability of draft and final deliverables and other milestones, and completion of each activity
b. An activity network, depicting sequential relationships and dependencies among activities and identifying those activities that impose the greatest time restrictions on the project
7. Project organization and resources.
This section shall be divided into the following paragraphs to describe the project organization and resources to be applied in each build.
7.1 Project organization.
This paragraph shall describe the organizational structure to be used on the project, including the organizations involved, their relationships to one another, and the authority and responsibility of each organization for carrying out required activities.
7.2 Project resources.
This paragraph shall describe the resources to be applied to the project. It shall include, as applicable:
a. Personnel resources, including:
1) The estimated staff loading for the project (number of personnel over time)
2) The breakdown of the staff-loading numbers by responsibility (for example, management, software engineering, software testing, software configuration management, software product evaluation, software quality assurance)
3) A breakdown of the skill levels, geographic locations, and security clearances of personnel performing each responsibility
b. Overview of developer facilities to be used, including geographic locations in which the work will be performed, facilities to be used, and secure areas and other features of the facilities as applicable to the contracted effort.
c. Acquirer-furnished equipment, software, services, documentation, data, and facilities required for the contracted effort. A schedule detailing when these items will be needed shall also be included.
d. Other required resources, including a plan for obtaining the resources, dates needed, and availability of each resource item.
8. Notes.
This section shall contain any general information that aids in understanding this document (e.g., background information, glossary, rationale). This section shall include an alphabetical listing of all acronyms, abbreviations, and their meanings as used in this document and a list of any terms and definitions needed to understand this document.
A. Appendixes.
Appendixes may be used to provide information published separately for convenience in document maintenance (e.g., charts, classified data). As applicable, each appendix shall be referenced in the main body of the document where the data would normally have been provided. Appendixes may be bound as separate documents for ease in handling. Appendixes shall be lettered alphabetically (A, B, etc.).

Appendix F Section 804 - Improvement of Software Acquisition Processes

This appendix addresses Section 804 of the Bob Stump National Defense Authorization Act for fiscal year 2003, Improvement of Software Acquisition Processes.

(a) ESTABLISHMENT OF PROGRAMS. -

(1) The Secretary of each military department shall establish a program to improve the software acquisition processes of that military department.

(2) The head of each Defense Agency that manages a major defense acquisition program with a substantial software component shall establish a program to improve the software acquisition processes of that Defense Agency.

(3) The programs required by this subsection shall be established not later than 120 days after the date of the enactment of this Act.

(b) PROGRAM REQUIREMENTS. - A program to improve software acquisition processes under this section shall, at a minimum, include the following:

(1) A documented process for software acquisition planning, requirements development and management, project management and oversight, and risk management.

(2) Efforts to develop appropriate metrics for performance measurement and continual process improvement.

(3) A process to ensure that key program personnel have an appropriate level of experience or training in software acquisition.

(4) A process to ensure that each military department and Defense Agency implements and adheres to established processes and requirements relating to the acquisition of software.

(c) DEPARTMENT OF DEFENSE GUIDANCE. - The Assistant Secretary of Defense for Command, Control, Communications, and Intelligence, in consultation with the Under Secretary of Defense for Acquisition, Technology, and Logistics, shall-

(1) Prescribe uniformly applicable guidance for the administration of all of the programs established under subsection (a) and take such actions as are necessary to ensure that the military departments and Defense Agencies comply with the guidance; and

(2) Assist the Secretaries of the military departments and the heads of the Defense Agencies to carry out such programs effectively by-

(A) Ensuring that the criteria applicable to the selection of sources provides added emphasis on past performance of potential sources, as well as on the maturity of the software products offered by the potential sources; and

(B) Identifying, and serving as a clearinghouse for information regarding, best practices in software development and acquisition in both the public and private sectors.

(d) DEFINITIONS. - In this section:

(1) The term "Defense Agency" has the meaning given the term in section 101(a)(11) of title 10, United States Code.

(2) The term "major defense acquisition program" has the meaning given such term in section 139(a)(2)(B) of title 10, United States Code.

Appendix G Embedded Systems Software Estimation Process

1.0 Background

This appendix provides detailed guidance for a software development estimating process that can be applied for several purposes, including 1) prior to RFP release in conjunction with the Integrated Risk Management Assessment (IRMA), 2) during source selection when estimates of proposed development schedules and associated effort are required, and 3) during the execution of the development program in conjunction with special reviews or Independent Review Teams (IRTs). When used in conjunction with IRTs, the process is used to estimate the required schedule and effort, provide credit for work accomplished, and derive an Estimate To Complete (ETC) in terms of schedule and effort. The process described herein is primarily oriented toward support of source selection.

Accurate, dependable, and convincing estimates must be developed to facilitate realistic program planning and to set reasonable expectations. This is best accomplished as a joint effort between the government and industry at the earliest possible time, starting prior to RFP release through discussions at pre-solicitation meetings (e.g., industry days). During source selection, every effort should be made (through discussions if necessary) to obtain the information required to support a comprehensive software development estimate.

For this process, the software development period is defined as starting with a baseline (complete) set of software requirements in a formal specification format and ending with a fully integrated and tested subsystem / functional software product ready for software / hardware integration and test. This convention is used because most of the estimating models provide estimates based on empirical data for only these phases. Additional effort is required to develop, allocate, and analyze the subsystem and software requirements; perform software to hardware (subsystem) integration and test; and perform system integration and test.

2.0 Process Overview

The software development estimating process consists of a series of activities that are grouped into the following process steps. These steps may be modified when working on estimates with SMC/FM:

a. Define the software structure
b. Identify all software to be developed
c. Determine software size information
d. Adjust for potential software size growth
e. Assess Commercial Off-The-Shelf (COTS) software and other reuse risk
f. Layout the software development approach
g. Select a set of consistent model input parameters
h. Estimate the development effort for each software component
i. Estimate the development schedule for each software component

j. Document the estimate assumptions
k. Present the software development estimate
l. Estimate the front-end systems engineering effort & schedule
m. Estimate the back-end systems integration and test
n. Develop a separate crosscheck for each estimate
o. Assess software schedule risk in the context of the Integrated Master Schedule (IMS)
p. Determine that the software development effort has been appropriately addressed throughout the proposal
q. Review the estimate with senior staff

3.0 Detailed Process Step Description:

a. Define the software structure - Starting with the planned weapon systems development description, determine (estimate) the software structure including major software subsystems and functions. For estimates during an on-going development, systems that have had a previous risk reduction phase such as Concept and Technology Development (C&TD), or systems in Source Selection, this information should be available from the developing contractor.

b. Identify all software to be developed - Using the software structure identified in step a, identify all software to be developed to at least the CSCI level if the phase of the program allows that level of detail. For early estimates before CSCIs have been established, major software subsystems (e.g. radar, flight controls, Global Positioning Systems) can each be assumed to be developed as a single interrelated set of software (i.e. radar could be identified as a single CSCI or a closely related set of CSCIs).

   When more detailed information regarding the software structure is available (e.g. during source selection) the estimates can be refined by estimating the refined software components separately and then aggregating them at the CSCI level. Be sure to include all software to be developed including both prime and subcontractor software development CSCIs. Lab simulations, Automated Test Equipment, Launch Vehicle Integration Support Facilities, System Integration Laboratories, Factory Test Equipment, and System/Software Engineering Environments are often as large or larger than the "operational" software, and often drive the development schedule.

c. Determine software size information - For each software component to be separately developed and hence individually modeled, determine the software size information. This information is required as an input to the software estimating models. For each separately modeled software component, decide how much software will be newly developed. Also determine how much existing software that is planned for incorporation into the system will be modified. Deletions from the existing software also need to be determined since there is a cost associated with deleting code from existing software. The ***Software***

Development Size and Schedule Summary or electronically submitted ***Software Parametric Data Input Form*** can be used to gather the data from the contractor through the proposal if the program is in source selection. This level of data is normally available only from the contractor or the offeror if in source selection. Estimate the data independently if contractor data is unavailable. EN home office functional engineering support is available to aid in the development of the software size estimate.

For rehosting (using modified or unmodified existing software), determine the percent re-design, re-code and re-test required. Do not assume that the re-test percentage is the same as the re-design percentage. A CSCI may have 10 percent of its design and code modified, however if that 10 percent is distributed across 50 percent of the CSCI's components, then the re-test percentage could be as high as 50 because 50 percent of the CSCI components would need to be re-tested. The SEER-SEM Software Cost Model provides a spreadsheet which determines re-design, re-implement (re-code), and re-test percentages from percentage change inputs such as Architectural Design, Documentation, Coding, Unit Test, Test Plans/Procedures/Reports, Test Drivers, Formal Test, etc. The electronic submittal parametric spreadsheet (***Software Parametric Data Input Form***) contains some additional inputs to help in this area such as percent CSUs modified and some breakdown of retest percentages. From a design and code perspective, be very careful not to credit beyond what is realistic. Experience in this area has consistently been that more reuse and adaptation is assumed up front than is realized during development. This is usually driven by the new system's performance requirements.

For incremental development, the estimates for re-design, re-code and re-test of existing software must be determined separately for each increment. For the first block of an incremental development the software to be reused includes only existing software from the baseline (existing) system. For subsequent blocks add the size of previous newly developed blocks to the existing baseline size. For example, assume that Block 1 is 50K Source Lines Of Code (SLOCs) with 10K of that reused from an existing system. Block 2 adds 25K SLOCs for a total of 75K SLOCS. For estimating block 2, 50K SLOCS (the product of block 1) is considered as reused software. Determine appropriate re-design, re-code and re-test for the prior block of software. These will more than likely be different than when the existing baseline software was reused during the previous block. The final block's re-test factor will probably be a large factor based on the need to test the total system against the original system requirement.
For incremental developments the software component sizing information must be allocated to each of the identified development blocks. The software sizing chart (***Software Development Size and Schedule Summary***) or parametric spreadsheet (***Software Parametric Data Input Form***) supports collecting data that allocates software size against the blocks as well as the start and end dates for each block. This data is required to derive an overall software development schedule including skewed block start dates.

Another consideration regarding reused software is that in many cases software is deleted. The impact of these deletions varies. If 10 percent of the existing code is deleted but it is distributed across many CSCIs or Computer Software Components (CSCs), the impact is greater than if the deleted 10 percent comprises one CSC. In the latter case, the re-test effort is significantly less.

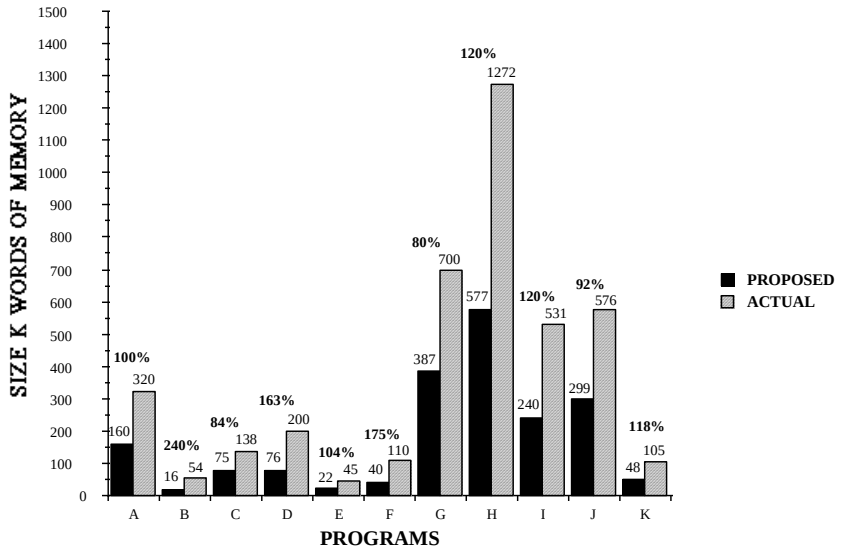
For systems that primarily use words of physical memory, function points, or objects as the measure of software size, determine the appropriate words per SLOC or SLOCs per function point/object conversion factors (for each language type and each CSCI).

Request the data using the sizing chart (***Software Development Size and Schedule Summary***) or the parametric spreadsheet (***Software Parametric Data Input Form***). Most of the models need the data in the format reflected in the sizing chart. Don't forget to include the changes to not only the operational software but also the support software as discussed above.

Once this data is obtained from the developer, evaluate the reasonableness and accuracy of the data provided using past experience on similar systems.

d. Adjust for potential software size growth

HISTORICAL SOFTWARE SIZE GROWTH



Adjust the software sizing information to account for anticipated software size growth over the development effort in terms of source lines of code. As the following chart illustrates, ASC programs have historically experienced up to 100 percent growth from the proposal phase to DT&E. Since this data was gathered, more rigorous requirements control and development processes have been

established along with an increased likelihood of precedent software. These factors may mitigate this level of growth on new programs. However, some level of growth should be estimated based on the program's characteristics. For example, unprecedented or unstable performance requirements or the failure of a COTS product to perform satisfactorily would increase the estimated growth percentage. For modifications to existing systems, actual size growth data may be available from previous system modifications.

e. Assess Commercial Off-The-Shelf (COTS) software and other reuse risk
It is appropriate to take a broad view of COTS to include all non-developmental software, because the risks are similar whether or not it is a commercial product. Types of software to consider include:

- True COTS software, shrink-wrapped and ready to go
- COTS software which needs to be tweaked through characterization data or minor modifications
- Non-developmental software which is not finished (i.e. developed outside program)
- Government furnished or co-developed software
- The aspects of COTS risk that can impact the software cost and schedule assessment include:
- Proposed COTS that does not fully support system requirements
- COTS integration problems with the rest of the system
- COTS which has substantial unneeded functionality
- COTS which does not support schedule needs

f. Layout the software development approach
Identify the software components that can start simultaneously and be developed concurrently. Identify software components that are dependent on other events including completion of system or software requirements definition and allocation or other software and hardware component activity completions. Defining these interdependencies will determine how the software development staffing profile is estimated as well as the critical path development for scheduling purposes. In the case of established interdependencies, appropriate offsets in the CSCI start dates must be incorporated in the overall schedule definition. It is critical to insure these issues are fully addressed in the Integrated Master Schedule (IMS). The quality of some past proposal IMS submittals in this regard has been a disappointment.

g. Select a set of consistent model input parameters
For the types of systems developed at SMC, three cost models have seen routine use over the years by both government and contractors: SEER, PRICE S, and REVIC (or other COCOMO based models). Information on these models is available at the Air Force Cost Analysis Agency (http://www.saffm.hq.af.mil/afcaa/). Brief descriptions of these tools follow:

SEER: The SEER (Systems Evaluation and Estimation of Resources) is a family of models consisting of parametric tools that can be used to estimate hardware development, software size, software development and integrated circuit development. SEER is Windows based and is very user friendly. Galorath Associates, Inc publishes SEER. AFMC has purchased a command wide license for SEER.

PRICE S: (Parametric Review of Information for Costing and Evaluation-Software) a parametric software-estimating tool originally developed by RCA's Government Systems Division. Subsequently owned by GE, Martin Marietta and now Lockheed, but still operating as a separate business entity. Widely used by the financial community. Previously very expensive, ran on mainframe computer, and recently ported to windows. This is a robust model with a user-friendly interface but complicated parameter entry for the occasional user. AFMC has purchased a command wide license for PRICE-S.

REVIC: The REVIC (Ray's Enhanced Version of Intermediate COCOMO) model predicts the development life-cycle effort for software development from requirements analysis through completion of software acceptance testing (FQT), and the maintenance life cycle for 15 years. Front-end software requirements and back-end software/hardware integration and system test effort not empirically derived; user must input percentage of baseline effort number to estimate total effort. REVIC software implementation is publicly available but not at all user friendly. Developed by Ray Kyle as an Air Force reservist. REVIC is the property of the US Air Force, and is currently available from AFCAA at: http://www.saffm.hq.af.mil/afcaa/models/REVIC92.EXE.

COCOMO: COCOMO (Constructive Cost Model) is an open public model developed and documented by Dr. Barry Boehm in his book "Software Engineering Economics". The model provides a set of basic equations that calculate the effort (person months) and schedule (elapsed time in calendar months) to perform a typical software development project based on an estimate of the lines of code to be developed and a description of the development environment. COCOMO originally was a manual process with paper and calculator and not implemented in a software model but has been used as the basis for many software cost models over the years. In 2000, a follow-on book titled “Software Cost Estimation with COCOMO II” was released, this time with an executable software program that implements the model.

Both the SEER and PRICE-S Models have been updated frequently and are currently well supported both with available training and comprehensive technical support. REVIC, however, has not been updated in the last 10 years.

There are many other software cost models that can be used successfully and nothing in this document is meant to discourage the use of these either as a supplement to or in lieu of the above-mentioned models.

These models require similar inputs, consisting primarily of software size information as well as a number of adjustment factors that are used to characterize the development, such as personnel capability, programming language, tools and process capability, criticality, and other factors.

The models can be extremely sensitive to the settings of some input parameters, and this requires care on the part of the model user to avoid severely misleading and inaccurate results. A set of model input parameters must be established, consistent across the models, which reflects the program specific characteristics, and developer's capability/tools, if known. If the developer is not known, choose input parameters normalized to the particular application (domain), e.g. avionics, flight control, simulator, or ATE. These parameters should be selected by a consensus of available software experts including stakeholders on the program, senior technical advisors, and where available the sustaining ALC perspective.

Where model input parameters are submitted by a bidder, care should be taken to make sure these inputs do not stray significantly from the nominal or expected domain normal values without substantial justification. Two problems have been noted in the past:
Bidders tend to adjust parameters to enhance their price competitiveness or perhaps just make mistakes in setting parameters. In several cases, inputs have been received for safety critical CSCIs with parameters set substantially differently.
In setting parameters relating to team experience, the current bid team who are generally more experienced are often used as the reference instead of the team expected in the midst of the software development phase.

This chart represents the input parameters from the REVIC model. Other models have similar but more extensive input parameters.

Typical Model Input Parameters (REVIC)	
Analyst Capability	Product Reliability
Programmer Capability	Data Base Size
Applications experience	Product Complexity
Virtual Machine Experience	Required Reuse
Programming Language Experience	Modern Programming Practices
Execution Time Constraint	Use of Software Tools
Main Storage Constraint	Required Security
Virtual Machine Volatility	Management Reserve for Risk
Computer Turnaround Time	Required schedule
Requirements Volatility	Software Development Mode

In order to select the model input parameters, an estimate of the percent completion of the system and software requirements definition, allocation and analysis needs to be established. The existing models treat requirements definition, allocation and analysis differently. In some, the requirements analysis is included in the requirements definition and allocation. In other models it is

accounted for separately. Take this into account when interpreting the model outputs. Most of the models will include in the estimate, but presented separately, the effort and schedule required to develop the software requirements. Some of the models also allow the estimator to include the effort required to maintain the software requirements after completion of the software requirements baseline.

The "core" estimate of virtually all of the models assumes that the software requirements are complete at the start of the development. As a result, if the complete software requirements baseline is not available, it is very important to include, in the estimate, the effort to define, allocate and analyze the software requirements. Software requirements analysis is treated differently between the models. Some models include this effort under software requirements definition, while other account for it separately. As a result, determine the percent completion of the software requirements analysis if at all possible.

Another input parameter area requiring definition relates to software to hardware integration and test. Most of the models will prompt the estimator for a percentage of the "core" software development effort to apply to the software to hardware integration and test. Be sure to include effort required for both software to hardware integration and test (subsystem integration and test) as well as subsystem-to-subsystem integration and test (system integration and test). The SEER model professes to estimate this effort from empirical data, Galorath Associates stated that the model's empirical data represents Space and Ground based weapon systems that are applicable to SMC systems.

h. Estimate the development effort for each software component

Each of the models requires the estimator to enter the input parameters as discussed above. In addition the models will prompt for the software structure or hierarchy including, at the lowest levels, the software components identified by the estimator for separate estimation. Some of the models allow the estimator to input a baseline set of input parameters for the project as a whole and then only modify particular parameters for each component. This saves a significant amount of time.

The output will be a collection of estimates (one for each component in the identified software structure). Each output will generally have three phases included in the estimate; 1) the software requirements phase, 2) the core software development phase (SRS to Software FQT), and 3) the software to hardware and subsystem integration and test phase. See the discussion below concerning the requirements and system integration phases. If the estimate you are performing assumes that the requirements and system integration phases will be included, then include the effort for them in the baseline estimate. If not, account for them as add-on estimates to the core software. Make sure different

model outputs are reporting the same categories of labor, e.g. engineering, test, CM, QA, management. If not, normalize them to a common baseline.

Effort estimates should be prepared using one of the models and crosschecked with another model, rule of thumb, or other estimation method. Ranges of reasonable input and parametric values should be established, which can be used to statistically generate a range of results from aggressive to conservative confidence levels. In the interim, these organizations should be contacted to assist in using the models in this prescribed manner. If the results from multiple models or different estimation techniques vary significantly, the estimating methods should be compared in detail. Where significant differences are noted, the reasons behind these differences should be understood through discussion with those who are familiar with the characteristics and shortcomings of the methods involved. Based on these discussions, an intelligent decision can be made on how to portray the estimation results and associated risks.

The estimate should be presented in terms of ranges (for both effort and schedule). In order to determine the range estimate, apply a range of confidence levels or cumulative probability (normally 50-90 percent). If multiple models are being used as in the examples below, each model is run twice for each software component to be estimated, as illustrated below using 50 & 90 percent confidence runs. Some models use the variance in the input parameters (lowest, most and highest likely) to compute the different percentage confidence estimates using a Monte Carlo simulation. This avoids having to run the model twice. The average of the lower confidence (aggressive) estimates represents the minimum range, and the average of the higher confidence (conservative) estimates represents the maximum range. In the example below, an analogy method was used as a crosscheck, which resulted in the model1 estimate being thrown out for subsystem 7 after agreement among those familiar with the model in question. Note that care must be taken to understand and define the model input parameters such that the results are reasonable.

		50%			90%		RANGE	
SYSTEM	MODEL1	MODEL2	Analogy	MODEL1	MODEL2	Analogy	MIN	MAX
SUBSYSTEM 1	1435.5	827.8	425.0	2009.7	1324.5	637.5	1131.7	1667.1
SUBSYSTEM 2	75.3	96.3	86.1	105.4	154.1	129.2	85.8	129.8
SUBSYSTEM 3	311.2	151.1	183.9	435.7	241.8	275.9	231.2	338.7
SUBSYSTEM 4	537.0	315.9	260.4	751.8	505.4	390.6	426.5	628.6
SUBSYSTEM 5	179.7	155.5	80.5	251.6	248.8	120.8	167.6	250.2
SUBSYSTEM 6	1278.6	1175.5	1187.0	1790.0	1880.8	1780.5	1227.1	1835.4
SUBSYSTEM 7	~~93.4~~	26.0	19.9	~~130.8~~	41.6	29.9	26.0	41.6
TOTALS	3910.7	2748.1	2242.8	5475.0	4397.0	3364.2	3295.7	4891.4

If the software is to be developed incrementally (blocks), use the following method to estimate the effort:

- Determine what portion of each component of software will be developed in each block. Often blocks of software are tied to numbered aircraft. For example, aircraft T-1 (first test aircraft) will likely include partial functional capability (flight controls, electrical management etc.), whereas aircraft T-8 (eighth test aircraft) will include the full avionics and weapons suite. Contractor planning information is usually required for this level of detail. If such information is not available, assume a realistic distribution of functionality across the blocks as described above. An alternative is to assume that each software component is developed incrementally in each increment. Assign a percentage of each software component to each block (e.g. 30/30/40 percent respectively for three blocks).

- Estimate the effort for each software component for each increment as above. When running the model for each increment, include only that portion of each CSCI allocated to the block being estimated. The result will be a collection of model runs as described above for each of the increments.

SUM OF MIN/MAX METHOD		
	EFFORT	
	MIN	MAX
INCREMENT 1	225	275
INCREMENT 2	495	605
INCREMENT 3	315	385
TOTAL	1035	1265

- Determine the minimum and maximum effort ranges for each increment, as described above using aggressive and conservative estimates (e.g., 50-90% confidence) and then summing the minimum and maximum ranges to arrive at a minimum and maximum range for the total development, as illustrated to the right.

i. Estimate the development schedule for each software component

The inputs and model runs required to estimate the effort in the previous step also provide a schedule estimate for each estimated software component. Just as in estimating effort, the schedule estimate should be prepared as a composite range (50-90 confidence) based on an appropriate combination of the model results.

Confidence range method as described above for effort is illustrated below:

- The key difference relative to schedules is that the individual component schedule estimates are not summed, as they were for effort, to arrive at a total. Each of the software components may be developed with some degree of parallelism (can be developed simultaneously), subject to dependencies that should be described in the Integrated Master Schedule so that generally the component with the longest schedule will drive the overall estimate. For example, if component X requires 42 months development time, component Y requires 36 months and component Z requires 38 months, then the overall schedule is still 42 months assuming the components are sufficiently independent. The important consideration is whether the individual component schedules support the system or sub-system development schedules, and that the longest component schedule supports the overall system development schedule.

- If the software is to be developed incrementally (in blocks), the same approach is used as for the effort estimate. The portion of the individual software components allocated to each block is included in each blocks estimate as illustrated below for blocks 1 and 2.

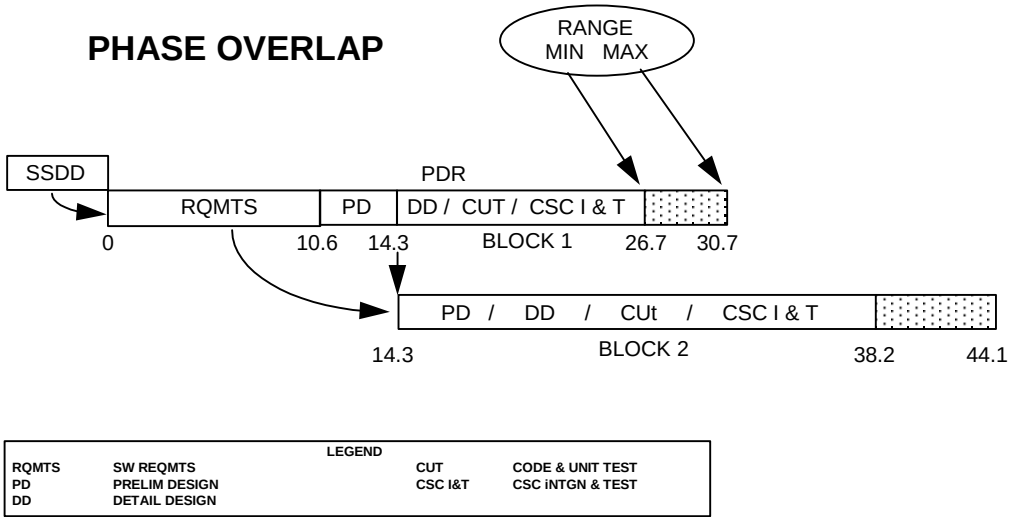
Schedules for block 1

				RANGE	
SYSTEM	MODEL1	MODEL2	Analogy	MIN	MAX
SUBSYSTEM 1	22.4	15.1	12.0	18.8	22.5
SUBSYSTEM 2	08.0	07.4	06.5	07.7	09.2
SUBSYSTEM 3	12.5	08.8	08.5	10.7	12.8
SUBSYSTEM 4	16.1	10.8	11.0	13.5	16.1
SUBSYSTEM 5	13.5	08.0	11.0	10.8	12.9
SUBSYSTEM 6	15.2	11.7	06.0	13.5	16.1
SUBSYSTEM 7	09.3	05.4	03.5	07.4	08.8

Schedules for block 2

SCHEDULE (MONTHS) BLOCK2					
				RANGE	
SYSTEM	MODEL1	MODEL2	Analogy	MIN	MAX
SUBSYSTEM 1	18.5	12.2	13.5	15.4	18.4
SUBSYSTEM 2	8	6.2	6.5	7.1	08.5
SUBSYSTEM 3	10.3	7.8	8.5	9.05	10.9
SUBSYSTEM 4	16.1	10.8	13.2	13.45	16.1
SUBSYSTEM 5	14.3	9.2	11	11.75	14.1
SUBSYSTEM 6	15.2	11.7	8	13.45	16.1
SUBSYSTEM 7	8.5	5.4	3.5	6.95	08.3

The end result is an estimate for each block that must then be realistically laid out in sequence. Just as before, the increments must be analyzed to determine if they can start simultaneously and run in parallel with no end to start dependencies. This is also done for the components internal to each block. Once the individual block schedules are determined, a reasonable overlap of blocks needs to be established (not all the blocks can start simultaneously). One traditionally used guideline that is supported in the literature and in practice is to not overlap the block design phases. For example, do not the start block 2 preliminary design phases until the block 1 preliminary design phase is completed (see illustration below). The same ideal can be applied to the other phases. Each activity completed in a previous block forms the baseline for starting the same activity for the subsequent block.



j. Document the estimate assumptions

Whenever an estimate is presented, the corresponding assumptions supporting the estimate should be documented to baseline the conditions under which the estimate was made. This documentation should include a description of what's included in the estimate; e.g., labor categories, development phases, software components, the state of requirements definition, assumptions for software size growth, personnel qualifications, etc. This allows the program office to re-accomplish an equivalent estimate in case the program characteristics change during the development and no longer fit the assumptions associated with the original estimate.

k. Present the software development estimate

In addition to presenting the schedule estimate as described, additional methods as describe below are available for formatting the results for presentation.

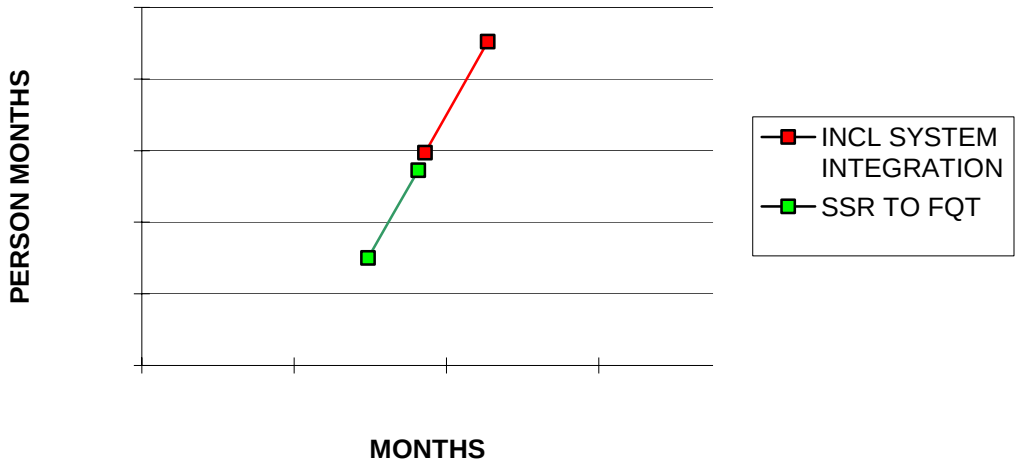
Applying the individual software component schedule durations into the overall program master level development schedule can create an overall development schedule. The impact of incremental development can also be reflected in the overall system development schedule. In that case, develop the increment's (block's) schedule using the block's individual components as described above. Then, at the system level, show the individual increments and their relation to the front-end requirements definition and allocation phases as well as the back-end system integration and test phases. See sections l and m for further information.

As discussed earlier, the schedule and effort estimates should be presented in terms of ranges rather than a single point value.

Present the estimating results in graphical form whenever possible. Some suggested methods of preparing and presenting these estimates are presented here.

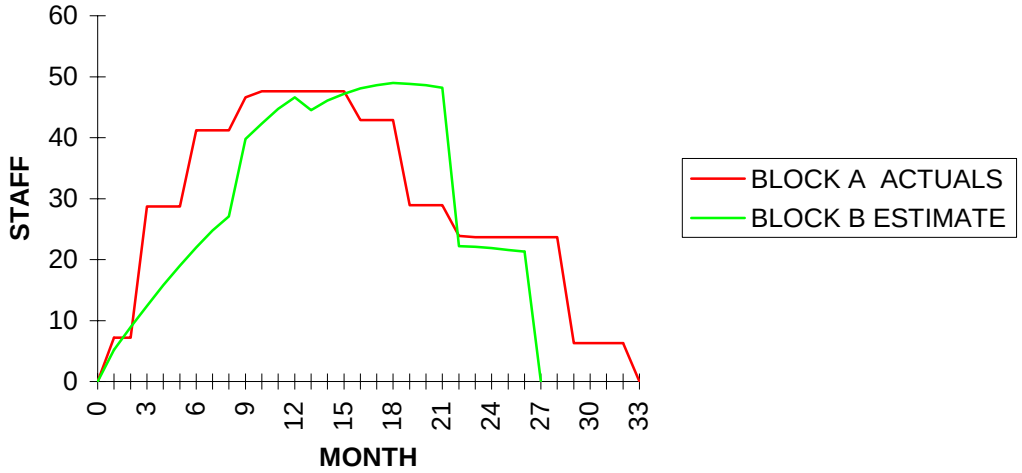
As illustrated below, a combined effort (person months) and schedule (months) range of estimates can be shown to include the basic software development period (SSR to Software FQT) and the total development including the system integration period. The range of estimates for each case is based on the estimating models with 50% to 90% cumulative probability to establish a range of expected results.

EFFORT/SCHEDULE RANGE



The figure illustrated below is an appropriate way to present the staffing profile over the schedule. Compare the estimated staffing to actuals where available. The example estimated the same contractor on the same program contrasts staffing profile shown here with the actual staffing profile of a similar completed block. Each estimating model will present the staffing profile in a different way. Some will actually generate a profile as illustrated below. Others provide the data in tabular fashion (by month).

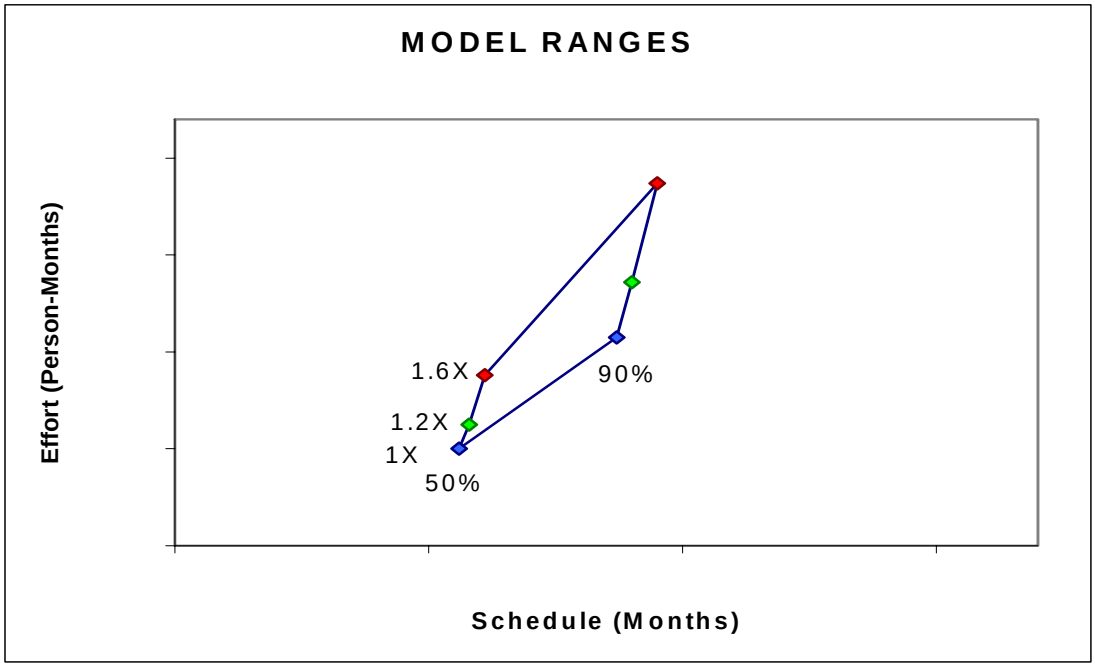
BLOCK A ACTUALS VS. BLOCK B ESTIMATE



The recommended method to establish the profile is to use the SEER-SEM model, which contains a sophisticated Raleigh curve based manpower profile generator and calibrate it such that effort and schedule prediction is close to the 90% confidence level. The danger is that the profile will be established only upon what one model estimates the profile to be. Consequently it makes sense to tabulate the 90% confidence level profile data from each model in a spreadsheet, check for any unusually large differences, and seek explanations from those familiar with each model. The goal is to establish an accurate expected spending profile for the software development so that it can be checked against the program funding profile and budget projections to prevent unexpected schedule slips if software development personnel cannot be brought on the program as needed.

The following illustration presents another method of presenting software development effort and schedule estimates in terms of ranges. Two point ranges are shown (50/90 cumulative probability spread horizontally) for three software size estimates. The two point ranges are developed using composites of the chosen models and 50-90 cumulative probability levels of schedule and effort. The 1X, 1.2X and 1.6X are three software estimates based on different size estimates. Size estimates are based on a nominal estimate (1X), 20 percent growth estimate (1.2X) and 60 percent growth estimate (1.6X). One would only use this approach if a growth factor were not already accounted for in the basic

size estimate. The chart presents an envelope within which the model estimates the program's effort and schedule will fall.

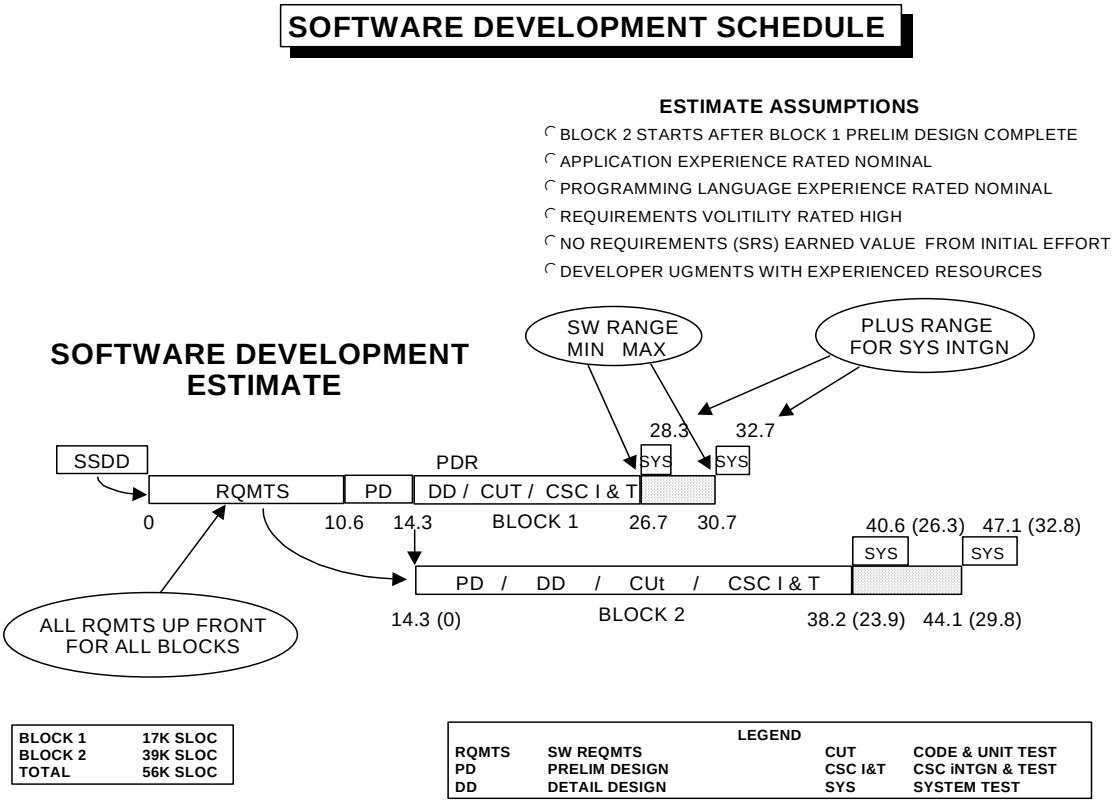


l. Estimate the front-end systems engineering effort & schedule

Estimate the software effort support to the up-front system engineering process (system and software requirements definition and allocation). The front-end effort should cover the effort through and including preparation of the approved Software Requirements Specifications (SRSs) or equivalent. This should include fully defined verification requirements (Section 4) and full allocation of all functional requirements to each planned functional increment (block). Most of the available models do not empirically derive the requirements definition effort. Although some have some empirically derived estimates, most prompt for (or have as a default) a percentage of the overall software development effort that represents the software requirements effort. Any historical data that can be obtained from the contractor will improve the requirements estimating process. The front-end software related systems engineering effort would be the sum of the individual requirements development effort estimates for each estimated software component. The longest software requirements duration from the individually estimated software components represents the overall duration of the software requirements definition phase.

For incrementally developed software, the software requirements definition should be developed up front for the entire development (all increments), just as in a non-incremental development. This scopes the development and prevents requirements instability or creep. Do not allow the requirements to be

incrementally developed at the start of each block. To model this up-front requirements effort for the entire incremental development effort, run the model(s) with the software components in a non-incremental fashion. That is to say, estimate the effort and duration for each component as if that component's development was not spread across multiple blocks. Do this for each component and then sum the efforts for the various software components to arrive at the total up-front software requirements effort. The model runs will also provide schedule estimates for this overall requirements effort (Largest of the software component efforts represents the duration of the total).

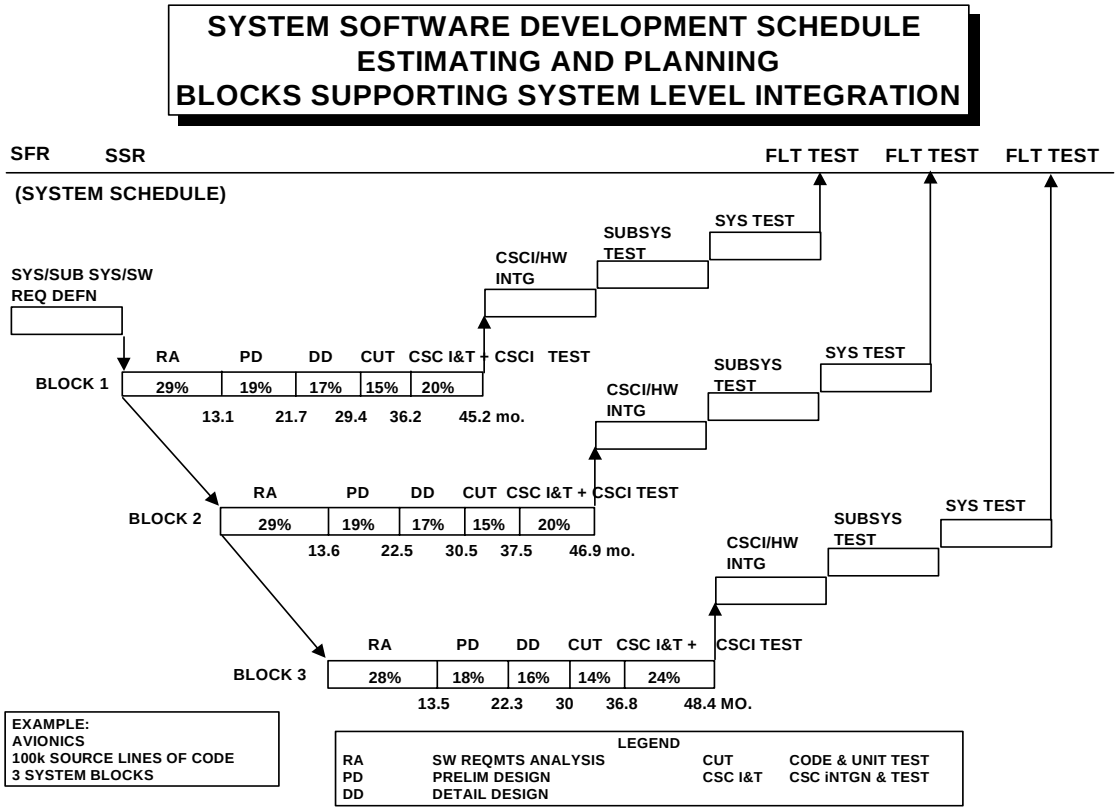


m. Estimate the back-end systems integration and test

The models take the same approach to the back-end system integration effort as they do for the front-end systems requirements estimation. Some models empirically derive the estimate but most use an add-on percentage of the basic software development effort.

As illustrated below, if incremental software development is used, the difference is the impact of regression testing (re-testing) of prior blocks as each block is integrated into the system. Block 2 incorporates all the functionality that was

tested in block 1 plus the functionality that was added in block 2. Therefore, a certain percentage of the software developed in block 1 will need to be re-tested during block 2 tests. The same applies for block 3. This impact will have been accounted for in the software integration and test (by adjusting the re-test parameter), but it also needs to be included at the system and subsystem level.



n. Develop a separate crosscheck for each estimate

Once the primary estimate has been developed using a parametric model, as discussed previously, it is a good policy to crosscheck the estimate using one or more additional models, a rule of thumb, or another estimation method. Using this approach eliminates the sole reliance on any single model whose results could be skewed for a particular set of program inputs. For example, some models provide significantly higher effort estimates for incorporation of existing software into the systems than do others. Using multiple models and methods normalizes these peculiarities.

o. Assess software schedule risk in the context of the Integrated Master Schedule (IMS)

One typical approach is to conduct Monte-Carlo schedule risk simulations. This is normally accomplished using a proposed IMS that is limited by RFP requirements to a certain number of tasks numbering in the hundreds. Inputs to support these simulations are normally supplied by engineering using schedule assessment sheets for each of the tasks on or close to the schedule critical path. In order to support these IMS assessments successfully, the RFP must request sufficient software schedule information to understand how the software development fits into the submitted IMS.

Typically the RFP-limited IMS will not contain sufficient information to identify each CSCI software development and each incremental delivery in support of block or spiral development. Without that information, a proper determination of software development schedule risk cannot be reliably accomplished. This problem can be overcome by specifying supplemental information in the RFP to be submitted with the proposal such as:

- IMS supplements to detail the lower level tasks for CSCI and incremental delivery
- Mapping information showing where each CSCI is addressed in the IMS

AX and/or FM personnel should determine schedule duration ranges for each CSCI, preferably using SEER-SEM. These schedule estimate results should be used to accomplish the following:

- Substantiate the placement of each CSCI development within the critical path
- Complete software task schedule assessment sheets using the aggressive (50%) and conservative (90%) estimates as the minimum and maximum, respectively
- Consider adjusting the most probable minimum or maximum range limits if the offeror's estimates differ, and ***strong*** justification is provided

p. Determine that the software development effort has been appropriately addressed throughout the proposal.

Establishing the completeness of the software portion of the bid should be a joint, coordinated effort between the program office, FM, AX, and PK participants in the source selection. FM and AX must jointly ensure that sufficient effort has been allocated for software development, and that the effort has been sufficiently scheduled and priced in the proposal. The PK role primarily concerns issues of COTS and non-developmental (a.k.a. free) software that is not charged to the program, to insure contractual coverage in case such software fails to support the contract requirements. This will require software to be broken out in the pricing at the CSCI level, including both prime and sub-contracted efforts.

Pricing justifications must include manpower estimates for all developmental software.

Some of the challenges in determining bid completeness include:

- Insufficient manpower planning in the proposal
- Disconnect between type of software developers proposed and associated rates
- Particular CSCI software development missing from bid through an oversight
- Insufficient manpower can result from bidders using 20% probability settings for effort in their supporting estimates, or from management challenges to their grass roots estimates to keep the overall bid low. The acquirer must also ensure all disciplines are included in the bid such as configuration management or quality assurance. Another pitfall to avoid is accepting low manpower estimates based on software productivity numbers derived from subsets of the necessary disciplines or software development phases and then ascribed to the entire job.

q. Review the estimate with senior staff

Review the final software effort and schedule estimate with the senior program managers and engineers. Review the estimate with the Source Selection Authority (SSA) or home office functionals if appropriate. Also, if appropriate, provide results to SMC/FM personnel to support budget submissions or software process improvement activities.

Appendix H References and Websites

Configuration Management

Software Technology Support Center Course: *Life Cycle Software Project Management*, Configuration Management, 9 October 2001.

Little Book of Configuration Management, Software Program Managers Network, November 1998. Download at: www.spmn.com/products_guidebooks.html

Program Manager's Guide for Managing Software, 0.6, 29 June 2001, Chapter 14: www.geia.org/sstc/G47/SWMgmtGuide%20Rev%200.4.doc

CM Today, Configuration management papers: www.cmtoday.com/yp/papers.html

Configuration Management Resource Guide: www.quality.org/config/CMResourceGuideMaster.doc

Configuration Management II Users Group: www.cmiiug.com

Crosstalk Magazine: www.stsc.hill.af.mil/crosstalk/

– "Software Configuration Management Terminology": www.stsc.hill.af.mil/crosstalk/1995/jan/terms.asp

– "Software Configuration Management: Function or Discipline?": www.stsc.hill.af.mil/crosstalk/1995/oct/cmfunct.asp

– "Stop-Gap Configuration Management": www.stsc.hill.af.mil/crosstalk/1998/feb/stopgapcm.asp

– "Effective Software Configuration Management": www.stsc.hill.af.mil/crosstalk/1998/feb/effectivecm.asp

– "Configuration Management Web Sites": www.stsc.hill.af.mil/crosstalk/1999/mar/cmsites.asp

– "Demystifying Software Configuration Management": www.stsc.hill.af.mil/crosstalk/1995/may/demystif.asp

– "Evaluation and Selection of Automated Configuration Management Tools": www.stsc.hill.af.mil/crosstalk/1995/nov/evaluati.asp

– "Software Configuration Management: A Discipline with Added Value": www.stsc.hill.af.mil/crosstalk/2001/jul/butler.asp

– "Introducing Process into Configuration Management": www.stsc.hill.af.mil/crosstalk/1996/jun/introduc.asp

– "Process-Based Configuration Management": www.stsc.hill.af.mil/crosstalk/1997/apr/configuration.asp

– "Achieving the Best Possible Configuration Management Solution": www.stsc.hill.af.mil/crosstalk/1996/sep/achievin.asp

– "A Tutorial on Control Boards": www.stsc.hill.af.mil/crosstalk/1999/mar/sorensen.asp

– "Adopting SCM Technology": www.stsc.hill.af.mil/crosstalk/1996/mar/adopting.asp

Department of Energy, Configuration Management resources: http://cio.doe.gov/sqse/pm_conf.htm

Institute of Configuration Management: www.icmhq.com

MIL-HDBK-61A, Configuration Management Guidance: www.assist2.daps.dla.mil/quicksearch/

Program Manager's Guide for Managing Software, 0.6, 29 June 2001, Chapter 14: www.geia.org/sstc/G47/SWMgmtGuide%20Rev%200.4.doc

Software Engineering Institute, Software Configuration Management support: www.sei.cmu.edu/legacy/scm/scmHomePage.html

Cost

Guide to the Project Management Body of Knowledge, A, Chapter 7, Project Management Institute, 2000.
Baker, Sunny and Kim, *Complete Idiot's Guide to Project Management, 2ed.*, Chapter 15, Alpha Books, 2000.
Chapman, James R., Cost Estimating, 1997, Principle Based Project Management website: www.hyperthot.com/project.htm
Flemming and Koppelman, "Earned Value Project Management, A Powerful Tool For Software Projects," Crosstalk, July 1998: www.stsc.hill.af.mil/crosstalk/1998/jul/value.asp
Air Force Cost Analysis Agency (AFCAA): www.saffm.hq.af.mil/afcaa/
Air Force Cost Reference Documents: www.saffm.hq.af.mil/afcaa/reference.html
Cost Tools: www.saffm.hq.af.mil/afcaa/models/models.html
Chapman, James R., Principle Based Project Management website: www.hyperthot.com/project.htm
Constructive Cost Model (COCOMO), information and software, University of Southern California, Center for Software Engineering: http://sunset.usc.edu/research/COCOMOII/index.html
Crosstalk Magazine: www.stsc.hill.af.mil/crosstalk/
– "Metrics Tools: Software Cost Estimation": www.stsc.hill.af.mil/crosstalk/1995/jun/metrics.asp
– "Cost Realism Methodology for Software-Intensive Source Selection Activities": www.stsc.hill.af.mil/crosstalk/1995/jun/cost.asp
– "Earned Value Project Management": www.stsc.hill.af.mil/crosstalk/1998/jul/value.asp
– "Pattern-Based Architecture: Bridging Software Reuse and Cost Management": www.stsc.hill.af.mil/crosstalk/1995/mar/pattern.asp
– "Does Calibration Improve Predictive Accuracy": www.stsc.hill.af.mil/crosstalk/2000/apr/ferens.asp
– "Project Recovery… It Can be Done": www.stsc.hill.af.mil/crosstalk/2002/jan/lipke.asp
– "Driving Quality Through Parametrics": www.stsc.hill.af.mil/crosstalk/1998/nov/galorath.asp
– "Future Trends, Implications in Cost Estimation Models": www.stsc.hill.af.mil/crosstalk/2000/apr/boehm.asp
– Practical Software Measurement, Performance-Based Earned Value": www.stsc.hill.af.mil/crosstalk/2001/sep/solomon.asp
– "New Air Force Software Metrics Policy": www.stsc.hill.af.mil/crosstalk/1994/apr/xt94d04a.asp
– "Statistical Process Control Meets Earned Value": www.stsc.hill.af.mil/crosstalk/2000/jun/lipke.asp
Guidelines for the Successful Acquisition and Management of Software-Intensive Systems (GSAM), Version 3.0, Chapter 13, OO-ALC/TISE, May 2000. Download at: www.stsc.hill.af.mil/gsam/guid.asp
MIL-HDBK-881, Work Breakdown Structure: www.acq.osd.mil/pm/newpolicy/wbs/mil_hdbk_881/mil_hdbk_881.htm

http://web2.deskbook.osd.mil/reflib/DDOD/003EH/001/003EH001DOC.HTM
NASA, *Parametric Cost Estimating Handbook, 2 ed*. Online version:
www.jsc.nasa.gov/bu2/PCEHHTML/pceh.htm
Download and print version: www.jsc.nasa.gov/bu2/NCEH/index.htm
Parametric Estimating Handbook:
http://web2.deskbook.osd.mil/reflib/DDOD/005EV/001/005EV001DOC.HTM#T2
Practical Software and Systems Measurement Support Center: www.psmsc.com
Software Cost Estimation Website: www.ecfc.u-net.com/cost/index.htm
Software Technology Support Center Course: *Life Cycle Software Project Management,*
Estimation, earned value,
etc., 9 October 2001.

COTS

Kohl, Ronald J., "COTS Based Systems: Benefits, Potential Risks and Mitigation Techniques":
www.geia.org/etmconf/Workshop/sed/COTS_Issues.pdf
"Opportunities and Complexities of Applying Commercial-Off-the-Shelf Components":
www.stsc.hill.af.mil/crosstalk/1998/04/applying.asp
"An Activity Framework for COTS-Based Systems":
www.stsc.hill.af.mil/crosstalk/2000/09/brownsword.html
- "The Double-Edged COTS IT Sword": www.stsc.hill.af.mil/crosstalk/1998/04/publisher.asp
- "Evaluating COTS Using Function Fit Analysis":
www.stsc.hill.af.mil/crosstalk/2000/02/holmes.html
- "A Web Repository of Lessons Learned from COTS-Based Software Development1":
www.stsc.hill.af.mil/crosstalk/2002/09/rus.html
"A COTS-Based Replacement Strategy for Aging Avionics Computers":
www.stsc.hill.af.mil/crosstalk/2001/12/haldeman.html
"The Commandments of COTS: Still in Search of the Promised Land":
www.stsc.hill.af.mil/crosstalk/1997/05/commandments.asp
"Lessons Learned From Using COTS Software on Space Systems":
www.geia.org/sstc/G47/SWMgmtGuide%20Rev%200.4.doc
SEI 1998 Software Symposium on COTS. Download slides:
www.sei.cmu.edu/cbs/cbs_slides/98symposium/

Metrics and Measurements

Perkins, Timothy K., "The Nine-Step Metrics Program," *Crosstalk* Magazine, February 2001:
www.stsc.hill.af.mil/crosstalk/2001/feb/perkins.asp
Augustine, Thomas, et al, "An Effective Metrics Process Model," *Crosstalk* Magazine, June 1999:
www.stsc.hill.af.mil/crosstalk/1999/jun/augustine.asp
Army Software Metrics Office, Computer Tutorials (under "Products"):
www.armysoftwaremetrics.org/
Crosstalk Magazine: www.stsc.hill.af.mil/crosstalk/
– "Metrics Tools: Effort and Schedule": www.stsc.hill.af.mil/crosstalk/1995/mar/metrics.asp
– "Project Recovery … It Can Be Done": www.stsc.hill.af.mil/crosstalk/2002/jan/lipke.asp
– "New Air Force Software Metrics Policy":
www.stsc.hill.af.mil/crosstalk/1994/apr/xt94d04a.asp

– "Universal Metrics Tools": www.stsc.hill.af.mil/crosstalk/1995/sep/universa.asp
– "Software Metrics Capability Evaluation Methodology and Implementation": www.stsc.hill.af.mil/crosstalk/1996/jan/metrics.asp
– "Metrics Tools": www.stsc.hill.af.mil/crosstalk/1995/feb/metrics.asp
– "Really Bad Metrics Advice": www.stsc.hill.af.mil/crosstalk/1998/aug/backtalk.asp
– "A Methodic Approach to Effective Metrics": www.stsc.hill.af.mil/crosstalk/1994/oct/xt94d10c.asp
– "Why the New Metrics Policy": www.stsc.hill.af.mil/crosstalk/1994/apr/xt94d04b.asp
– "Metrics: Problem Solved?": www.stsc.hill.af.mil/crosstalk/1997/dec/metrics.asp
– "Metrics Tools: Size": www.stsc.hill.af.mil/crosstalk/1995/apr/metrics.asp
– "Quantitative Software Management Workshop - Software Cost and Schedule Estimation": www.stsc.hill.af.mil/crosstalk/1996/oct/xt96d10h.asp
– "Software Quality Metrics for Object-Oriented Environments": www.stsc.hill.af.mil/crosstalk/1997/apr/quality.asp
– "Metrics for Predicting Run-Time Failures and Maintenance Effort": www.stsc.hill.af.mil/crosstalk/1998/aug/predicting.asp
– "Metrics for Ada 95: Focus on Reliability and Maintainability": www.stsc.hill.af.mil/crosstalk/1995/may/ada95.asp
– "Pro-Active Metrics": www.stsc.hill.af.mil/crosstalk/1998/aug/proactive.asp
– "Best Measurement Tool Is Your Telephone": www.stsc.hill.af.mil/crosstalk/2001/mar/lucero.asp
– "Metrics Tools: Software Cost Estimation": www.stsc.hill.af.mil/crosstalk/1995/jun/metrics.asp
– "Software Maintainability Metrics Models in Practice": www.stsc.hill.af.mil/CrossTalk/1995/nov/Maintain.asp
– "Earned Value Project Management": www.stsc.hill.af.mil/crosstalk/1998/jul/value.asp

Guidelines for the Successful Acquisition and Management of Software-Intensive Systems (GSAM), Version 3.0,
Chapter 13 & Appendix N, OO-ALC/TISE, May 2000. Download at:
www.stsc.hill.af.mil/gsam/guid.asp

Literate Programming Software Metrics, many resources:
www.literateprogramming.com/fmetrics.html

NASA Software Assurance Technology Center: http://satc.gsfc.nasa.gov/support/
– Recommended code metrics, by language:
http://satc.gsfc.nasa.gov/metrics/codemetrics/index.html

Requirements Management References

The Standish Group, "The Scope of Software Development Project Failures," 1995.

DoD, *Program Manager's Guide For Managing Software*, v. 0.6, Chapter 6, 21 June 2001.

Gause, Donald and Weinberg, Gerald, *Exploring Requirements: Quality Before Design*, Dorset House, 1989.

Software Technology Support Center Course: *Life Cycle Software Project Management*, Project Initiation, 9 October 2001.

Wiegers, Karl E., "When Telepathy Won't Do: Requirements Engineering Key Practices," Cutter IT Journal,

May 2000. www.processimpact.com/articles/telepathy.html
Raghavan, Zelesnik, & Ford, “Lecture Notes of Requirements Elicitation,” 1994: www.sei.cmu.edu/publications/documents/ems/94.em.010.html
Davis, Alan M., *Software Requirements Analysis and Specification*, Prentice-Hall, 1990.
Crosstalk Magazine: www.stsc.hill.af.mil/crosstalk/
– “Writing Effective Natural Language Requirements Specifications”: www.stsc.hill.af.mil/crosstalk/1999/feb/wilson.asp
– “Experiences in the Adoption of Requirements Engineering Technologies”: www.stsc.hill.af.mil/crosstalk/1998/dec/cook.asp
– “An Examination of the Effects of Requirements Changes on Software Releases”: www.stsc.hill.af.mil/crosstalk/1998/dec/stark.asp
– “Doing Requirements Right the First Time”: www.stsc.hill.af.mil/CrossTalk/1998/dec/hammer.asp
– “Four Roads to Use Case Discovery”: www.stsc.hill.af.mil/CrossTalk/1998/dec/ham.asp
– “An Examination of the Effects of Requirements Changes on Software Releases”: www.stsc.hill.af.mil/CrossTalk/1998/dec/stark.asp
– “Experiences in the Adoption of Requirements Engineering Technologies”: www.stsc.hill.af.mil/CrossTalk/1998/dec/cook.asp
– “Recommended Requirements Gathering Practices”: www.stsc.hill.af.mil/crosstalk/2002/apr/young.asp
– “Making Requirements Management Work for You”: www.stsc.hill.af.mil/crosstalk/1999/apr/davis.asp
Department of Energy (DOE) Software Engineering Methodology, Chapter 4: http://cio.doe.gov/sqse/sem_toc.htm
Department of Energy, Software Risk Management Practical Guide: http://cio.doe.gov/sqse/pm_req.htm
Department of Justice Systems Development Life Cycle Guidance, Chapter 6: www.usdoj.gov/jmd/irm/lifecycle/table.htm
IEEE Computer Society: http://computer.org
– 4th International Conference on Requirements Engineering (ICRE'00): http://computer.org/proceedings/icre/0565/0565toc.htm
INCOSE Requirements Working Group: www.incose.org/rwg/
– “Factors Influencing Requirement Management Toolset Selection”: www.incose.org/lib/rmtools.html
– “Characteristics of Good Requirements”: www.incose.org/rwg/goodreqs.html
– “Requirements Management Technology Overview”: www.incose.org/tools/reqsmgmt.html
– “What is a Requirement?”: www.incose.org/rwg/what_is.html
Guidelines for the Successful Acquisition and Management of Software-Intensive Systems (GSAM), Version 3.0,
Chapter 11, OO-ALC/TISE, May 2000. Available for download at: www.stsc.hill.af.mil/gsam/guid.asp
Heimdahl and Associates, “Requirements Verification Checklist”:

Risk Management

Software Technology Support Center Course: *Life Cycle Software Project Management,* Project Initiation, 9 October 2001.

Risk Management Guide for DoD Acquisition, Chapter 2, February 2001. www.dsmc.dsm.mil/pubs/gdbks/risk_management.htm

Higuera, Ron & Haimes, Yacov, "Software Risk Management," Technical Report, 1996. www.sei.cmu.edu/publications/documents/96.reports/96.tr.012.html

- *Risk Management Guide for DoD Acquisition,* Appendix B, February 2001.

- Arizona State University (ASU), "Question List for Software Risk Identification." www.eas.asu.edu/~riskmgmt/qlist.html

Department of Energy, Risk Assessment Questionnaire. http://cio.doe.gov/sqse/pm_risk.htm

Arizona State University software risk management resources: www.eas.asu.edu/~riskmgmt/

Guidelines for the Successful Acquisition and Management of Software-Intensive Systems (GSAM), Version 3.0, Chapter 6, OO-ALC/TISE, May 2000. Download at: www.stsc.hill.af.mil/gsam/guid.asp

Crosstalk Magazine: www.stsc.hill.af.mil/crosstalk/

– "A Practical Approach to Quantifying Risk Evaluation Results": www.stsc.hill.af.mil/crosstalk/2000/feb/hantos.asp

– "A Risk Management Bibliography": www.stsc.hill.af.mil/crosstalk/1994/mar/xt94d03k.asp

– "Continuing Risk Management at NASA": www.stsc.hill.af.mil/crosstalk/2000/feb/rosenberg.asp

– "Identifying and Managing Risks for Software Process Improvement": www.stsc.hill.af.mil/crosstalk/2000/feb/risk.asp

– "Managing Risk Management": www.stsc.hill.af.mil/crosstalk/1999/jul/neitzel.asp

– "Managing Risk with TSP": www.stsc.hill.af.mil/crosstalk/2000/jun/webb.asp

– "Risk Management in Practice": www.stsc.hill.af.mil/crosstalk/1997/apr/management.asp

– "Team Risk Management": www.stsc.hill.af.mil/crosstalk/1995/jan/teamrisk.asp

Department of Energy, Software Risk Management Practical Guide: http://cio.doe.gov/sqse/pm_risk.htm

Department of Justice, *Systems Development Life Cycle Guidance Document,* Risk Management, Chapter 3: www.usdoj.gov/jmd/irm/lifecycle/table.htm

Higuera, Dorofee, Walker, & Williams, "Team Risk Management: A New Model for Customer Supplier Relationships," 1994. Download at: www.sei.cmu.edu/publications/documents/94.reports/94.sr.005.html

Higuera, Ron, et. Al., *Continuous Risk Management Guidebook*, 1996, CMU.

Program Manager's Guide for Managing Software, 0.6, 29 June 2001, Chapter 5: www.geia.org/sstc/G47/SWMgmtGuide%20Rev%200.4.doc

Risk Management Guide for DoD Acquisition, 2001. Download at: www.dsmc.dsm.mil/pubs/gdbks/risk_management.htm

Risk Radar, Software for managing risk. Available at: www.iceincusa.com/rskrdr.htm

Software Engineering Institute, Risk management overview: www.sei.cmu.edu/programs/sepm/risk/

Software Engineering Institute , risk management frequently asked questions: www.sei.cmu.edu/programs/sepm/risk/risk.faq.html

US Treasury, Systems Development Life Cycle Handbook, Risk Management Processes, Chapter 5. Download:
www.customs.ustreas.gov/contract/modern/sdlcpdfs/tocsdlc.htm

Software Engineering

Software Engineering Institute (SEI):
- –A federally funded research and development center whose focus is to improve the state of software practice throughout the defense community
- –http://www.sei.cmu.edu/

Software Technology Support Center (STSC):
- –An Air Force organization that provides services and support to organizations responsible for software development and maintenance, including program offices and contractors
- –http://www.stsc.hill.af.mil/
- –*CROSSTALK*, the Journal of Defense Software Engineering Published monthly by the Air Force Software Technology Support Center

Download from http://www.stsc.hill.af.mil/

Information on the CMMI models can be found at:
- –http://www.sei.cmu.edu/cmmi/
- –The website also provides other information, such as CMMI frequently asked questions.

Information on Capability Maturity Models (including the CMMI and SW-CMM) can be found at:
- –http://www.sei.cmu.edu/cmm/cmms/cmms.html
- –The CMMI, SW-CMM model, and many SEI references can be downloaded from this website.

Publications on CMM, CBA-IPI, and SCE:
- –*The Capability Maturity Model for Software*, Version 1.1, (SEI-93-TR-25)
- –Paulk, M., et al, *The Capability Maturity Model — Guidelines for Improving the Software Process*, CMU/SEI, (SEI-93-TR-25), 1995, Addison-Wesley
- –*Software Capability Evaluation Version 3.0, Implementation Guide for Supplier Selection*, CMU/SEI-95-TR-012, April 1996
- –*Software Capability Evaluation Version 3.0, Method Description*, CMU/SEI-96-TR-002, April 1996
- –Dunaway, D., Masters, S., *CMM-Based Appraisal for Internal Process Improvement (CBA IPI): Method Description*, Software Engineering Institute, CMU/SEI-96-TR-007, Apr. 1996
- –*Process Maturity Profile: Software CMM CBA-IPI and SPA Appraisal Results 2002 Year End Update, April 2003–An Executive Introduction to CMM-Based Software Process Improvement*, 1999
- –Herbsled, et al, *Benefits of CMM-Based Software Process Improvement: Initial Results*, SEI-94-TR-013, Aug. 1994

Sustainment

Department of Energy (DOE) *Software Engineering Methodology*, Chapter 10:
http://cio.doe.gov/sqse/sem_toc.htm

Program Manager's Guide for Managing Software, 0.6, 29 June 2001, Chapter 12:
www.geia.org/sstc/G47/SWMgmtGuide%20Rev%200.4.doc

Caputo, Kim, *CMM Implementation Guide*, Addison Wesley Longman, Inc., 1998.

Crosstalk Magazine: www.stsc.hill.af.mil/crosstalk/
– "Confusing Process and Product: Why the Quality is not There Yet": www.stsc.hill.af.mil/crosstalk/1999/07/cook.asp
– "It's Time for ISO 9000": www.stsc.hill.af.mil/crosstalk/1994/03/xt94d03i.asp
– "Coding Cowboys and Software Processes": www.stsc.hill.af.mil/crosstalk/1997/08/processes.asp
– "Improvement Stages": www.stsc.hill.af.mil/crosstalk/1998/10/cusick.asp
– "The Process King vs. the Cowboys": www.stsc.hill.af.mil/crosstalk/1997/08/backtalk.asp
– "Using the CMM Effectively": www.stsc.hill.af.mil/crosstalk/1995/10/usingcmm.asp
– "If You Get Straight A's, You Must Be Intelligent - Respecting the Intent of the Capability Maturity Model": www.stsc.hill.af.mil/crosstalk/1998/02/respecting.asp
– "Software Process Proverbs": www.stsc.hill.af.mil/crosstalk/1997/01/proverbs.asp
– "Preparing for a CMM Appraisal": www.stsc.hill.af.mil/crosstalk/1996/08/preparin.asp
– "Ten Things Your Mother Never Told You About the Capability Maturity Model": www.stsc.hill.af.mil/crosstalk/1998/09/kulpa.asp
International Standards Organization, ISO 90001, www.iso.ch (also available from http://qualitypress.asq.org)
Jeffries, Ron, "Extreme Programming and the Capability Maturity Model": www.xprogramming.com/xpmag/xp_and_cmm.htm
Paulk, Mark C., "Effective CMM-Based Process Improvement": www.sei.cmu.edu/publications/articles/effective -spi.html
Randall's Practical Resources Online, ISO 9000 and Quality Related Topics: http://home.earthlink.net/~rpr-online/Contents.htm#ISO9000
Software Engineering Institute, CMMI and resources: www.sei.cmu.edu/cmmi/
Software Insight Tool, Checklists for acquisition and risk mitigation, etc.: www.sed.monmouth.army.mil/sit/sitstart.htm
TickIT information, www.tickit.org
University of Massachusetts Dartmouth, Software Process Resource Collection: http://www2.umassd.edu/SWPI/

Testing

Software Program Managers Network, *Little Book of Testing, Vol. 1*, 1998: www.spmn.com/products_guidebooks.html
Jorgensen, Paul C., *Software Testing A Craftsman's Approach*, CRC Press, 1995, p.3.
Kit, Ed, *Software Testing in the Real World*, Addison-Wesley, 1995, p.3.
Software Program Managers Network, *Little Book of Testing, Vol. II*, 1998: www.spmn.com/products_guidebooks.html
Program Manager's Guide for Managing Software, 0.6, 29 June 2001, Chapter 11: www.geia.org/sstc/G47/SWMgmtGuide%20Rev%200.4.doc
University of South Australia, Software Testing notes: http://louisa.levels.unisa.edu.au/se1/testingnotes/testing.htm
Crosstalk Magazine: www.stsc.hill.af.mil/crosstalk/
– "The Problem with Testing": www.stsc.hill.af.mil/crosstalk/2001/07/index.html
– "Maintaining the Quality of Black-Box Testing": www.stsc.hill.af.mil/crosstalk/2001/05/korel.html
– "Proven Techniques for Efficiently Generating and Testing Software":

www.stsc.hill.af.mil/crosstalk/2000/06/wegner.html
– “Model to Assess Testing Process Maturity”:
www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1998/11/burnstein.asp
– Planning Efficient Software Tests”:
www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1997/10/planning.asp
– “Using Statistical Process Control with Automatic Test Programs”:
www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1998/08/statistical.asp
– “Engineering Practices for Statistical Testing”:
www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1998/04/statistical.asp
– “Using Inspection Data to Forecast Test Defects”:
www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1998/05/inspection.asp
Department of Energy (DOE) *Software Engineering Methodology*, Chapters 7-9:
http://cio.doe.gov/sqse/sem_toc.htm
NASA, Recommended Approach to Software Development, Sections 7-9:
http://sel.gsfc.nasa.gov/website/documents/online-doc.htm

Appendix I Acronyms

ARC	Appraisal Requirements for CMMI
ASP	Acquisition Strategy Panel
CAIV	Cost as an Independent Variable
CARD	Cost Analysis and Requirements Description
CCaR	Comprehensive Cost and Requirement
CDD	Capability Development Document
CDR	Critical Design Review
CDRL	Contract Data Requirement List
CI	Configuration Item
CM	Configuration Management
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integrated
COCOMO	Constructive Cost Model
CONOPS	Concept of Operations
COTS	Commercial-off-the-shelf
CPU	Computer Processor Utilization
CRLCMP	Computer Resources Lifecycle Management Plan
CRWG	Computer Resources Working Group
CSCI	Computer Software Configuration Item
DAE	Defense Acquisition Executive
DID	Data Item Description
DoD	Department of Defense
DR	Discrepancy Report
EVMS	Earned Value Management System
FCA	Functional Configuration Audit
FFRDC	Federally Funded Research and Development Center
FQR	Formal Qualification Review
GFM	Government Furnished Materials
GFE	Government Furnished Equipment
GOTS	Government-off-the-shelf
HWCI	Hardware Configuration Item
ICD	Interface Control Document
IDD	Interface Design Document
IMP	Integrated Master Plan
IMS	Integrated Master Schedule
IPT	Integrated Product Team

IRS	Interface Requirements Specification
IRT	Independent Review Team
KPA	Key Process Area
LCC	Life Cycle Cost
NDI	Non-Developmental Items
NSA	National Security Agency
O&M	Operating and Maintenance
OCD	Operation Concept Description
OSD	Office of the Secretary of Defense
PCA	Physical Configuration Audit
PDR	Preliminary Design Review
PO	Project Officer
POE	Program Office Estimate
RA	Requirements Analysis
RFP	Request for Proposal
RMP	Risk Management Plan
SA	Software Acquisition
SAMP	Single Acquisition Management Plan
SCAMPI	Standard CMMI Method for Process Improvement
SCCB	Software Configuration Change Board
SCM	Software Configuration Management
SCMP	Software Configuration Management Plan
SDD	Software Design Document
SDP	Software Development Plan
S/SEE	System/Software Engineering Environment
SEI	Software Engineering Institute
SETA	System Engineering and Technical Advisor
SIV&V	Software Independent Verification and Validation
SLOC	Source Lines of Code
SMC	Space and Missile Systems Center
SOO	Statement of Objectives
SOW	Statement of Work
SPO	System Program Office
SPS	Software Product Specification
SQA	Software Quality Assurance
SQAP	Software Quality Assurance Plan/Process
SRS	Software Requirements Specification
SSR	Software Specification Review
SSS	System or System Segment Specification

STD	Software Test Description
STP	Software Test Plan
STR	Software Test Report
SU	Software Unit
SUM	Software Users Manual
SW	Software
SWCI	Software Configuration Item
SWEIT	Software Engineering Integration and Test
TEMP	Test & Evaluation Master Plan
TIM	Technical Interchange Meeting
TRR	Test Readiness Review
T&E	Software Test and Evaluation
VDD	Version Description Document
WBS	Work Breakdown Structure

This page intentionally left blank